GENERAL INSTRUMENT

CP1600/1610 MICROPROCESSOR DATA MANUAL

Microelectronics Division
General Instrument Corporation

GENERAL INSTRUMENT

CP1600/1610 MICROPROCESSOR DATA MANUAL

ARCHITECTURE

INSTRUCTION SET

SYSTEM IMPLEMENTATION

ELECTRICAL SPECIFICATIONS

SUPPORT

TABLE OF CONTENTS—PAGE 2

NOVEMBER 1982

©Copyright 1982 GENERAL INSTRUMENT CORPORATION
The information in this publication, including schematics, is suggestive only. General Instrument Corporation does not warrant, not will it be responsible or liable for, (a) the accuracy of such information, (b) its use or (c) any infringement of patents or other rights of third parties.

Table of Contents

1. INTRODUCTION
1.1 Description 5
1.2 Features 5

2. ARCHITECTURE
2.1 Basic Functional Blocks 6
2.2 Pin Assignments 7
2.3 Pin Functions 8
2.3.1 Data Bus 8
2.3.2 Bus Control 8
2.3.3 Processor Control 11
2.3.4 Power Supplies/Clocks 11
2.4 Processor Timing 12
2.5 State Timing 14
2.5.1 Initialization Sequence 16
2.5.2 Fetch Sequence 16
2.5.3 Address Sequence 16
2.5.4 Execute Sequence 17
2.5.5 Bus Request Sequence 17
2.5.6 Interrupt Sequence 17
2.6 Timing Diagrams 18

3. INSTRUCTION SET
3.1 General Instruction Format 31
3.1.1 Instruction Overview 31
3.1.2 Symbolic Notation 32
3.1.3 Operation of Status Bits 32
3.2 Internal Reference Instructions 33
3.2.1 Register to Register 33
3.2.2 Single Register 33
3.2.3 Register Shift 33
3.2.4 Internal Control 34
3.2.5 Jump/Jump and Save Return 34
3.3 External Reference Instructions 35
3.3.1 Data Access Instructions 35
3.3.2 Conditional Branch Instructions 36
3.3.3 External Reference Addressing Modes 36
3.3.4 Double Byte Data 37
3.4 Program Examples 38
3.4.1 Loop or Iteration Control 38
3.4.2 Table Access 39
3.4.3 Transfer via an Indexed Dispatch Table 40
3.4.4 Evaluate Variables and Transfer 40
3.4.5 Arithmetic Operations 40
3.4.6 Stack Operations 40
3.4.7 Interrupt Processing 41
3.4.8 Subroutines 42
3.4.9 BEXT Instruction (Polling) 44
3.5 Instruction Set (Summary Listing) 45
3.6 Instruction Set (Detailed Listing) 46
3.6.1 Register to Register 46
3.6.2 Single Register 47
3.6.3 Register Shift 47
3.6.4 Control 48
3.6.5 Jump 48
3.6.6 Branches 49
3.6.7 Direct Addressed Data—Memory 50
3.6.8 Immediate Data—Register 51
3.6.9 Indirect Addressed Data—Register 51
3.6.10 Immediate Double Byte Data—Register 52

3.6.11 Indirect Addressed Double Byte Data—Register 53
3.6.12 Symbolic Notation 53

4. SYSTEM IMPLEMENTATION

4.1 Bus Structure 54
4.2 Clocks 56
4.3 External Sense Logic 57
4.4 Start/Stop-Halt Functions 58
4.5 Interrupt System 59
4.6 Basic Input/Output Ports 60
4.7 Simple Communication Interface 61

5. ELECTRICAL SPECIFICATIONS

5.1 Maximum Ratings 62
5.2 Standard Conditions 62
5.3 DC Characteristics 62
5.4 AC Characteristics 63
5.5 Capacitance 64
5.6 Timing Diagram References 65
5.7 Package Outlines 68

List of Illustrations

Fig. 1 CP1600 Internal Block Diagram 6
Fig. 2 CP1600 Pin Assignments 7
Fig. 3 Bidirectional Bus Timing 12
Fig. 4 Pipeline Timing 13
Fig. 5 Microinstruction Format 13
Fig. 6 Bus Control Signals 14
Fig. 7 Simplified State Flow Diagram 15
Fig. 8 Initialization Sequence 19
Fig. 9 Instruction Fetch 20
Fig. 10a Instruction Fetch and Data Access (Direct) 21
Fig. 10b Instruction Fetch and Data Access (Indirect) 22
Fig. 11 Write Sequence 23
Fig. 12a Branch Sequence 24
Fig. 12b External Condition Test Sequence 25
Fig. 13 $\overline{\text{TCI}}$ Instruction Timing 25
Fig. 14 $\overline{\text{PCIT}}$ Timing 26
Fig. 15 $\overline{\text{Interrupt}}$ Sequence 27
Fig. 16 $\overline{\text{BUSRQ}}$/$\overline{\text{BUSAK}}$ Sequence 28
Fig. 17 BDRDY Timing 29
Fig. 18 Simple System Implementation 54
Fig. 19 Advanced System Implementation 55
Fig. 20 Typical Clock Generation Circuit 56
Fig. 21 16 Mutually Exclusive Branch Tests 57
Fig. 22 4 Test Conditions with Multiple "AND" Capability 57
Fig. 23 CP1600 Operational Mode Control/Indication 58
Fig. 24 Nested Interrupt Structure 59
Fig. 25a I/O Port—Example 1 60
Fig. 25b I/O Port—Example 2 60
Fig. 25c I/O Port—Example 3 60
Fig. 26 UAR/T Interface 61
Fig. 27 Clock and Bus Timing 65
Fig. 28 Master Sync Timing 65
Fig. 29 Bus Acknowledge Timing 66
Fig. 30 Interrupt Timing 66
Fig. 31 External Branch Timing 67
Fig. 32 40 Lead Dual In-line Packages 68

1 INTRODUCTION

1.1 Description

The CP1600/CP1610 are compatible members of the Series 1600 Microprocessor products family. Each is a complete, 16-bit, single chip, high speed MOS-LSI Microprocessor. The Series 1600 family is fabricated with General Instrument's N-Channel Ion-Implant GIANT II process, insuring high performance with proven reliability and production history. All members of the Series 1600 family are fully compatible with the CP1600/CP1610.

The Microprocessor has been designed for high speed data processing and real time applications. Typical applications include programmable TV games, home computer systems/home information centers, programmable calculator systems, peripheral controllers, process controllers, intelligent terminals and instruments, data acquisition and digital communications processors, numerical control systems and many general purpose mini-computer applications. The Microprocessor can readily support a variety of peripheral equipment such as TTY, CRT display, tape reader/punch, A/D & D/A converter, keyboard, cassette tape, floppy disk, and RS-232C data communication lines.

The CP1600/CP1610 utilize third generation minicomputer architecture with eight general purpose registers to achieve a versatile, sophisticated microcomputer system. The 16-bit word enables fast and efficient processing of alphanumeric or byte oriented data. The 16-bit address capability permits access to 65,536 words in any combination of program memory, data memory, or peripheral devices. This single address space concept, combined with a powerful instruction set, provides an efficient solution to microcomputer and many minicomputer-based product requirements.

1.2 Features

- ☐ 8 program accessible 16-bit general purpose registers
- ☐ 86 basic instructions
- ☐ 4 addressing modes: direct, immediate, indirect, relative
- ☐ Conditional branching on status word and 16 external conditions
- ☐ Unlimited interrupt nesting and priority resolution
- ☐ 16-bit logic and 2's complement arithmetic
- ☐ Status word: carry, overflow, sign, zero
- ☐ Direct memory access (DMA) for high speed data transfer
- ☐ 64K memory using single address
- ☐ TTL compatible/simple bus structure
- ☐ CP1600: 600ns cycle time, 3.3 MHz 2-phase clock
- ☐ CP1610: 1μs cycle time, 2 MHz 2-phase clock

2 ARCHITECTURE

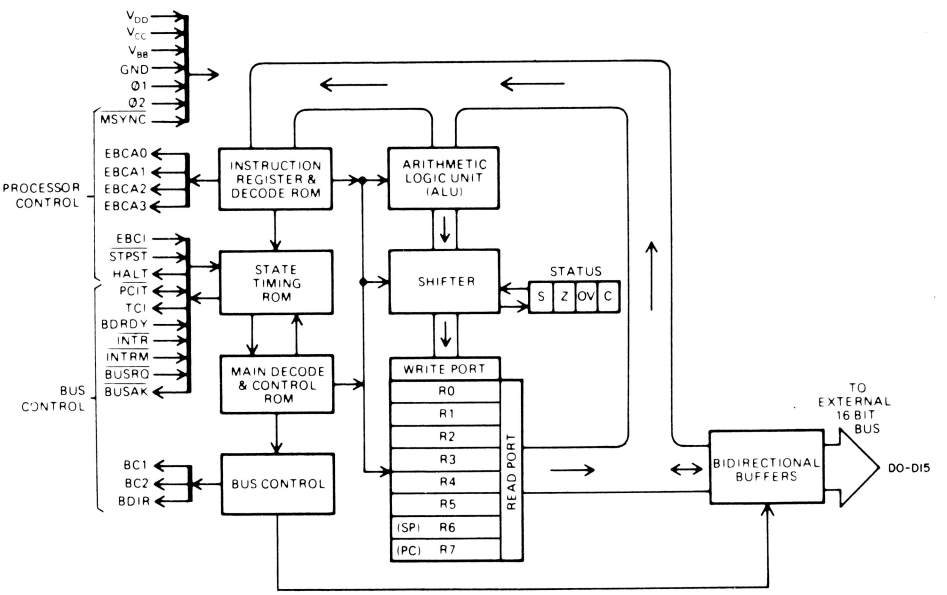
The CP1600 is a general register oriented, central processing unit utilizing eight 16-bit registers. These registers may be used as accumulators or as addressing pointers to locations external to the CPU. R7 and R6 have special characteristics which distinguish them from the other registers. R7 serves as the program counter (PC) and points to the next instruction to be executed. R6 serves as a stack pointer (SP) which points to the next available location in the "last-in, first-out" stack maintained anywhere in main memory.

2.1 Basic Functional Blocks

A basic internal block diagram of the CP1600 showing major subsystems and data flow is shown in Figure 1. It is noted that all data transfers within the machine are performed on 16-bit words processed in two 8-bit bytes. Internally, the architectural effect of this byte serial technique is to organize all data paths as 8 bits in width. In communicating with external devices, however, the CPU transfers 16 bits in parallel.

The major logic blocks through which all data flow are the Arithmetic Logic Unit (ALU), Shifter, Register Array, and the Internal 16-bit Bus.

Fig. 1 CP1600 INTERNAL BLOCK DIAGRAM



The ALU is the primary data processing element of the CPU. It accepts two data words each 8 bits wide from the internal 16-bit bus and produces an 8-bit resultant.

The ALU passes the resultant to the Shifter which is also 8 bits wide. The Shifter can transfer data to the register array transparently or can skew data one bit position right or left. The Shifter in combination with the ALU can effect changes in the four status bits which monitor CARRY OUT from the most significant bit of the resultant, ARITHMETIC OVERFLOW, SIGN DETECT, and ZERO DETECT.

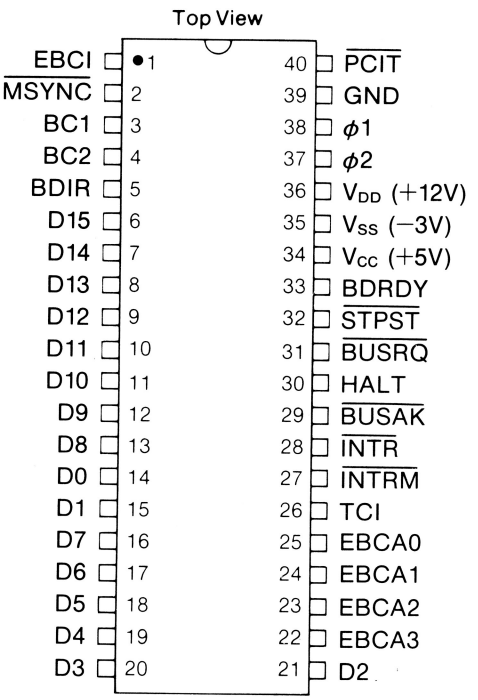
The Register Array is comprised of the eight 16-bit registers with one 8-bit write port and two 8-bit read ports. The write port can direct data to any one of the eight registers and then either to that register's lower or upper 8-bit byte. The read ports can simultaneously output to the internal bus any two byte combination, be they from the same register, different registers, left or right byte.

The Internal 16-bit Bidirectional Bus is the primary link for all information transfers between the CPU and the external world. Communications between the internal bus and the external bus are governed by the bus control signals BC1, BC2 and BDIR.

2.2 Pin Assignments

The CP1600 is supplied in a 40 lead dual in-line package with the pin assignments as shown in Fig. 2.

Fig. 2 CP1600 PIN ASSIGNMENTS



2.3 Pin Functions

2.3.1 DATA BUS

D0-D15 (input/output/high impedance): pins 6-21
Data 0-15

These 16 signals comprise the 16-bit bidirectional bus used by the CPU to send both data and addresses to the external subsystems and receive data and instructions from the external subsystems, i.e., memory, printers, keyboards, displays, A/D and D/A converters, magnetic tape, cassettes, and other peripherals of various types. All operations on the bus involve 16-bit transfers of either data, address or instruction words.

2.3.2 BUS CONTROL

BDIR (output): pin 5
Bus DIRection

This signal indicates the direction of the 16-bit bidirectional bus at the CPU. A logic "1" (High) indicates that D0-D15 are ouputting to the bus. A logic "0" (Low) indicates that D0-D15 are either inputting from the bus or are in a high impedance state.

BC2, BC1 (output): pins 4, 3
Bus Control 2, 1

These signals in conjunction with BDIR are used to control all external bus operations. In simple systems, they can be used directly as control signals while in more elaborate ones they can be decoded in a single 3 to 8 decoder to provide all eight external bus control functions listed below:

BDIR	BC2	BC1	BUS CONTROL	FUNCTION
0	0	0	NACT	NO ACTION. This signal indicates the CPU is not using the bus. D0-D15 are in the high impedance state.
0	0	1	ADAR	ADDRESSED DATA TO ADDRESS REGISTER. This signal causes the addressed contents of memory to be gated onto the bus and strobed into the address register. It is activated during all instructions which specify direct addressing. D0-D15 are in the high impedance state.
0	1	0	IAB	INTERRUPT ADDRESS TO BUS. This signal which occurs during the INTERRUPT SEQUENCE of the CPU, serves to gate the starting address of the interrupt service routine for the acknowledged interrupting device onto the bus. D0-D15 are in an input mode. The CPU will input this starting address into the PC to commence execution of the service routine. IAB also occurs during the power-up initialization sequence to input the starting address of the main program to the PC.

BDIR	BC2	BC1	BUS CONTROL	FUNCTION
0	1	1	DTB	DATA TO BUS. This signal is used to gate data from memory or any peripheral device onto the bus. Data in this sense can mean instructions, addresses, or data. D0-D15 are in an input mode.
1	0	0	BAR	BUS TO ADDRESS REGISTER. This signal is used to load the contents of the bus into the external address register. D0-D15 are in an output mode.
1	0	1	DW	DATA WRITE. This signal is functionally identical to DWS except that it occurs one machine cycle time prior to DWS. It is used for extended write operation purposes. D0-D15 are in an output mode.
1	1	0	DWS	DATA WRITE STROBE. This signal is used as a write enable for memory or any peripheral device. D0-D15 are in an output mode.
1	1	1	INTAK	INTERRUPT ACKNOWLEDGE. This signal denotes CPU acceptance of an interrupt and is used by the interrupting devices to resolve priority. The INTERRUPT SEQUENCE of the internal CPU state timing logic commences with this acknowledgement. Further details of the interrupt service are given in Section 2.5.6. D0-D15 are in an output mode.

$\overline{\text{BUSRQ}}$ (input): pin 31
BUS ReQuest

This signal is low active and causes the CPU to relinquish all control of the bus to allow other devices to carry out direct bus transfer operations. The CPU grants use of the bus only after the completion of an interruptable instruction (refer to the instruction listings for those instructions which are interruptable and those which are not). The CPU signifies release of the bus by causing $\overline{\text{BUSAK}}$ to go low and the CPU will remain in this condition until the external device releases $\overline{\text{BUSRQ}}$.

$\overline{\text{BUSAK}}$ (output): pin 29
BUS AcKnowledge

This is a low active signal used to inform external devices that the bus has been released by the CPU in response to the $\overline{\text{BUSRQ}}$ signal.

BDRDY (input): pin 33
Bus Data ReaDY

This signal is used to make the CPU "wait" and resynchronize itself to peripheral subsystems that cannot respond to requests for reads and writes at full CPU speed. This is intended primarily for synchronization to slow speed memories. The duration of the "wait" period must be less than 40 microseconds to preserve the dynamic state of the CPU.

2.3 Pin Functions (cont.)

$\overline{\text{INTR}}$, $\overline{\text{INTRM}}$ (inputs): pins 28, 27
INTerrupt Request
$\overline{\text{INTerrupt Request Maskable}}$

These two signals are low active signals that request the CPU to honor an interrupt at the completion of any interruptable instruction under the following conditions.

a) $\overline{\text{INTR}}$ is always honored by the CPU and hence is the highest priority interrupt request line.

b) $\overline{\text{INTRM}}$ is honored by the CPU only if the internal CPU interrupt flip-flop, INTFF, is set. This flip-flop is controlled via the EIS and DIS instructions and several others that have similar capability as described in the instruction set.

The CPU will enter the INTERRUPT SEQUENCE if either $\overline{\text{INTR}}$ is active or if $\overline{\text{INTRM}}$ is active and INTFF is set. During this sequence, the CPU will store the PC onto the top of the memory stack and resolve interrupt device priority by enabling INTAK. This is followed by inputting the starting address of the interrupt service routine from the appropriate interrupting device into the PC by enabling IAB. The next instruction is fetched from the location pointed to by the new PC and execution of the service routine begins from that point.

$\overline{\text{PCIT}}$ (input/output): pin 40
$\overline{\text{Program Counter Inhibit/Trap}}$

This pin provides two functions.

a) As an input, this signal is a low active signal that prevents the incrementation of the program counter (R7) during the fetch phase of all instructions. Care should be used when utilizing this function during multiword instructions.

b) As an output, this signal will generate a low active pulse during the execution of the SIN (Software INterrupt) instruction. Properly used, this signal can be returned to the CPU by the interfacing hardware as an interrupt request on either of the $\overline{\text{INTR}}$ or $\overline{\text{INTRM}}$ input pins. The interrupt will be acknowledged at the end of the SIN instruction.

These functions will not interact with each other under normal operation. The Timing Diagrams in Section 2.6 will precisely describe the timing of each function.

TCI (output): pin 26
Terminate Current Interrupt

This high active pulse signal is generated via the TCI instruction to terminate the highest priority interrupt presently in service.

2.3.3 PROCESSOR CONTROL

EBCA0, EBCA1, EBCA2, EBCA3 (outputs): pins 25-22
External Branch Condition Address 0, 1, 2, 3,

These lines are the buffered outputs from the 4 least significant bits of the instruction register (bits 0-3) and are used to externally select one of sixteen digital states to be sampled by the CPU during the execution of the BEXT (Branch on EXTernal) instruction.

EBCI (input): pin 1
External Branch Condition In

This is the return signal of the 1 of 16 selection made by EBCA 0, 1, 2, 3. It is high active such that a logic "1" (High) would test as true and cause the branch to occur as part of the BEXT instruction. A logic "0" would not allow a branch to occur.

$\overline{\text{MSYNC}}$ (input): pin 2
Master $\overline{\text{SYNC}}$

This signal is used to synchronize the internal clock generator logic to the main $\phi 1$, $\phi 2$ clock phases. Synchronization is achieved by remaining active (low) for at least ten milliseconds after power is applied to the CPU and becoming inactive (high) on any rising edge of a $\phi 1$ clock. That $\phi 1$ clock corresponds to an internal TS3. This signal also disables the interrupt system.

$\overline{\text{STPST}}$ (input): pin 32
$\overline{\text{S}}$ToP **$\overline{\text{S}}$T**art

This is an edge-triggered signal used to control the running condition of the CPU. If the CPU is presently running, the negative transition of $\overline{\text{STPST}}$ will cause the CPU to stop but only after the completion of an interruptable instruction. The CPU will generate a high active HALT signal acknowledging the stopped mode. The next negative transition of $\overline{\text{STPST}}$ will cause the CPU to return to the run mode. The HALT output will then return to a logic "0" (Low) condition.

HALT (output): pin 30

This is a high active signal indicating that the CPU is in the stopped mode. This mode can occur by either the toggle action from the $\overline{\text{STPST}}$ input or by the execution of a HALT instruction by the CPU.

2.3.4 POWER SUPPLIES/CLOCKS

V_{DD}: pin 36
Nominal +12 Volt power supply input to the CPU.

V_{CC}: pin 34
Nominal +5 Volt power supply input to the CPU.

V_{BB}: pin 35
Nominal −3 Volt power supply input to the CPU.

GND: pin 39
Ground reference.

$\phi 1$, $\phi 2$ (inputs): pins 38, 37

These signals supply the high-level, high-speed, non-overlapping two phase clocks to the CPU.

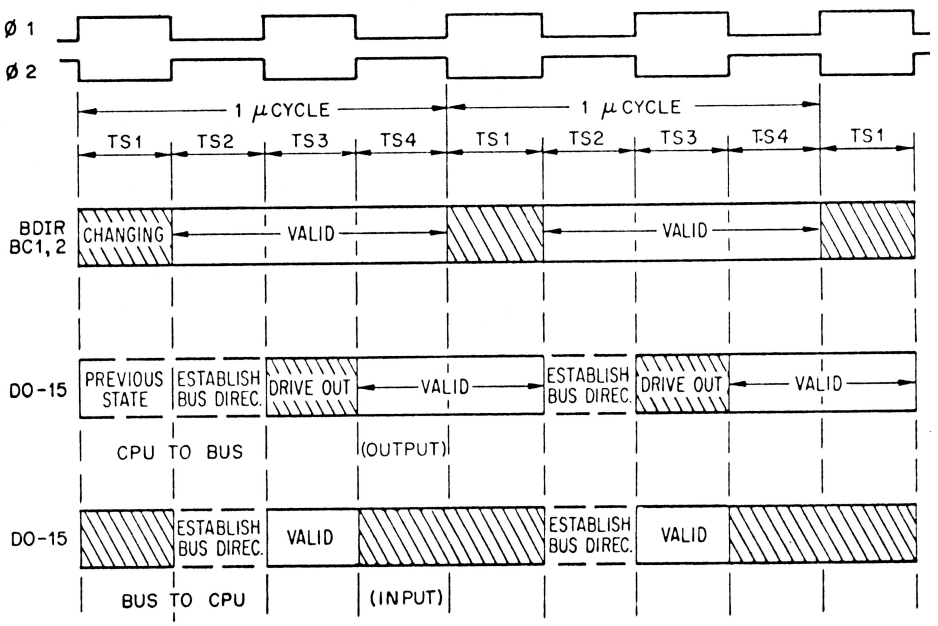
2.4 Processor Timing

The sequential operation of the CPU is governed by an internal State Timing ROM. This network generates next addresses to the Main Decode and Control ROM. It maps these addresses from the Instruction Register contents and makes next address decisions based on conditions in the machine and/or the present microinstruction. The Main Decode and Control ROM generates 21-bit microinstructions during each processor cycle. These microinstructions are grouped into various Fetch states. Address states, Execute states, Wait states, states which control the Interrupt and Bus Request Sequences, and others. Each micro state time is defined as four internal time slots—TS1, TS2, TS3, TS4—as generated by the Clock Generator from the external non-overlapping clocks. See Figure 3.

If the CPU is driven by a 3.3 MHz, two-phase clock system as shown, then the duration of each micro state is 600 nanoseconds.

The State Timing ROM operates in parallel with the data processing logic thus keeping the Control ROM synchronous with the data flow operations. Thus, during any internal state time, the Control ROM and the Instruction Register Decode ROM are dictating the data processing functions in the ALU, Shifter, and the Register Array.

Fig. 3 BIDIRECTIONAL BUS TIMING



The CP1600 achieves high internal processing speed through the use of microinstruction overlapping and data pipelining (Fig. 4). During each 600 nanosecond cycle, a 21-bit control word is executed while the next address decision is made and the new control word accessed. In addition, the Register/ALU data loop is pipelined so that two sets of byte pairs are being processed during each cycle. Before the result of an ALU operation is complete and returned to the Registers, the next operation on new data has already started.

In order to achieve this "data streaming" effect, the microinstruction word of the processor is divided into three major fields (Fig. 5). The first controls the top of the pipe which includes the READ PORTS and front end of the ALU logic. The second controls the bottom of the pipe which includes the lookahead carry network, the final part of the ALU logic and shifter, and the rewrite to the registers. The third field controls the selection of the next control state.

Fig. 4 PIPELINE TIMING

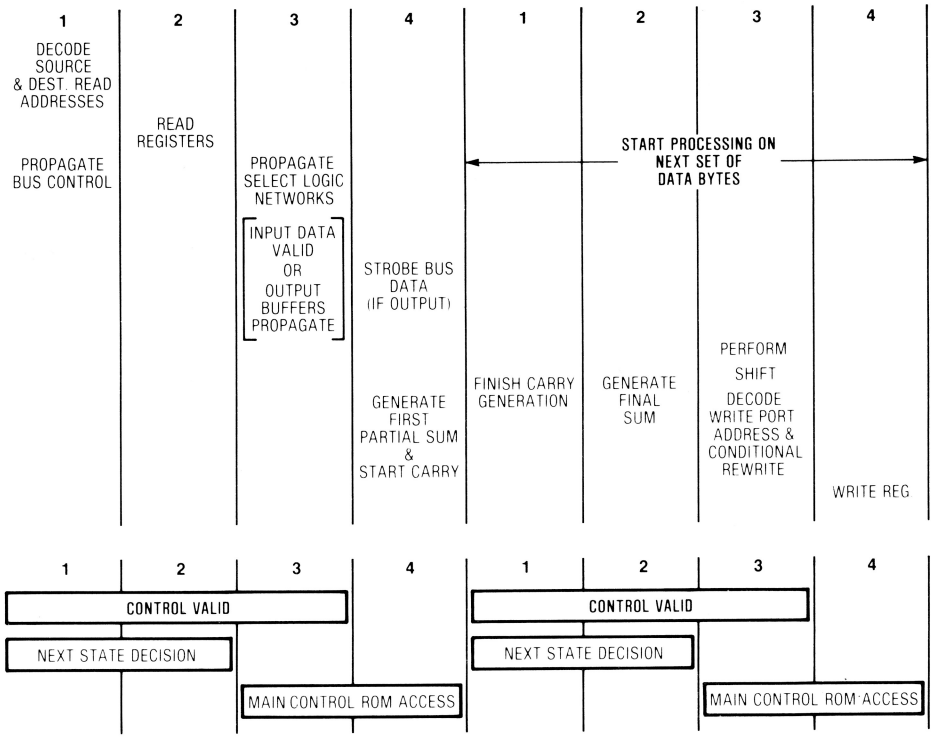
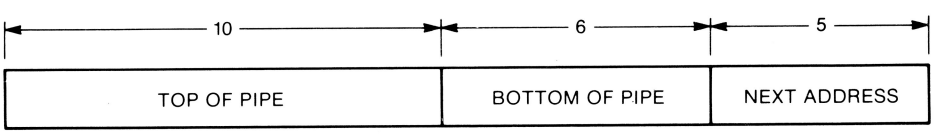


Fig. 5 MICROINSTRUCTION FORMAT



2.5 State Timing

Although it is not necessary for the user to understand the exact sequencing through the internal microcoded states, it is important to understand the general state flow and how it relates to the execution of instructions and their interaction with the various interfacing signals like BDRDY, $\overline{\text{BUSRQ}}$, $\overline{\text{INTR}}$ and others. A basic state flow diagram is shown in Fig. 7. Illustrations of the exact timing during each of these sequences is given in the Timing Diagrams in Section 2.6.

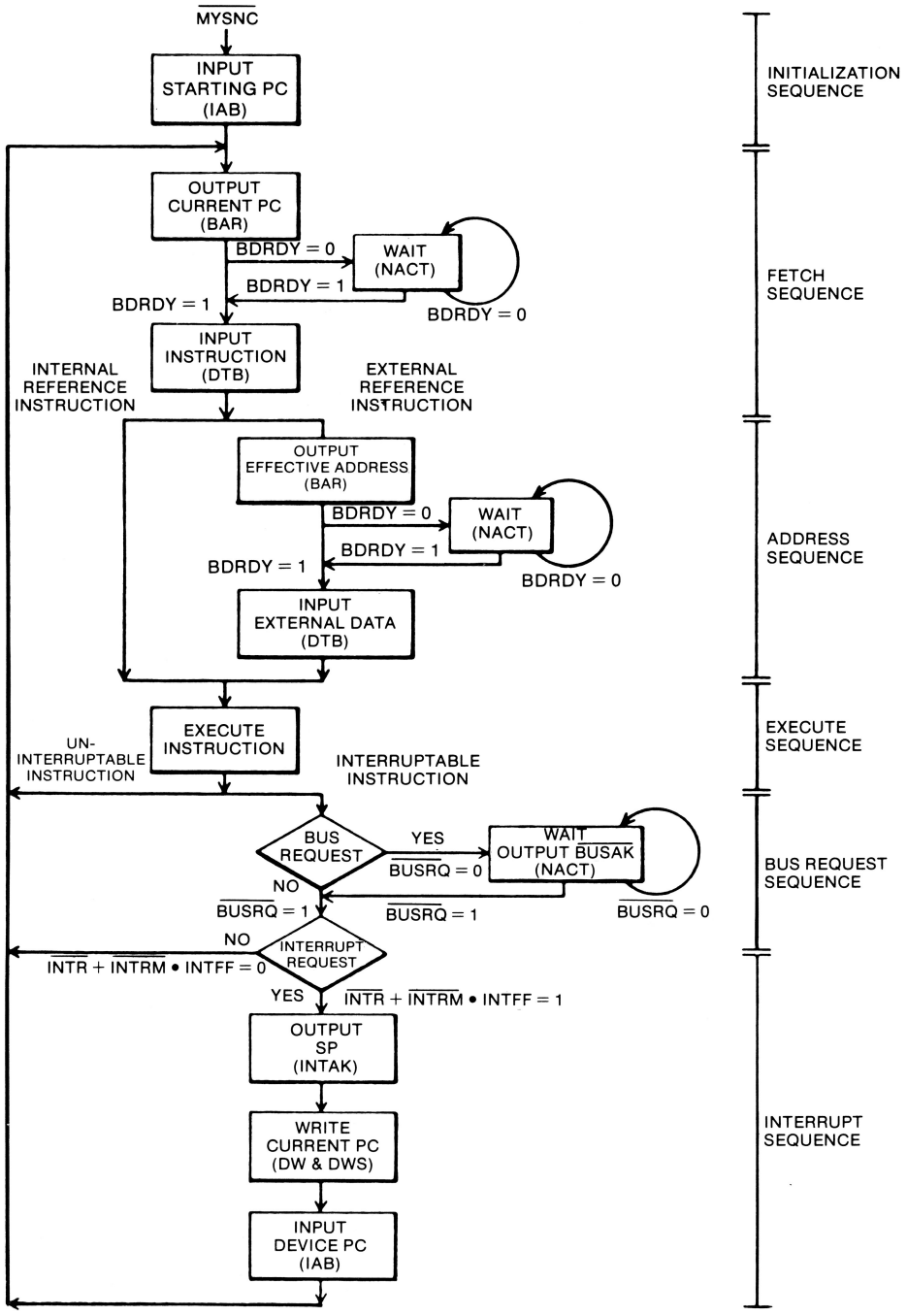
There are six major state sequences through which the CPU passes: Initialization, Fetch, Address, Execute, Bus Request, and Interrupt sequences. Each of the sequences consist of one or more micro states (indicated by rectangular blocks). However, the decoded bus control signal in parentheses indicates only the primary bus function to be performed during that functional operation. Recall that each microcode state has its own unique pattern of bus control signals (BDIR, BC2, BC1). For example, in the first fetch state illustrated in Fig. 7, the bus controls signals will be decoding BAR. In the next, they decode NACT, and, in the next, they decode DTB. Fig. 6 summarizes these decoded functions.

Fig. 6 BUS CONTROL SIGNALS

BDIR	BC2	BC1	Signal	Decoded Function
0	0	0	NACT	No ACTion, D0-D15 = high impedance
0	0	1	ADAR	Address Data to Address Register, D0-D15 = high impedance
0	1	0	IAB	Interrupt Address to Bus, D0-D15 = Input
0	1	1	DTB	Data To Bus, D0-D15 = Input
1	0	0	BAR	Bus to Address Register
1	0	1	DW	Data Write
1	1	0	DWS	Data Write Strobe
1	1	1	INTAK	INTerrupt AcKnowledge

INPUT STARTING (IAB)

Fig. 7 SIMPLIFIED STATE FLOW DIAGRAM



2.5 State Timing (cont.)

2.5.1 INITIALIZATION SEQUENCE

When power is first applied to the CPU, the $\overline{\text{MSYNC}}$ signal must be at an active (low) level. This will force the CPU to an inactive state in which D0-D15 are in a high impedance state with the bus control signals issuing NACT. All other signals are in their inactive state and the interrupt system is disabled. All eight internal registers and the status bits contain arbitrary, undefined data. When $\overline{\text{MSYNC}}$ goes inactive (high), the bus control signals issue IAB, and the CPU inputs from the bus into the PC the starting address of the main program. Note that the initialization address can be defined by the user at any desired bus address or can be the default address resulting from the logical state of the non-driven bus.

2.5.2 FETCH SEQUENCE

The Instruction Fetch sequence will be entered after the initialization of the CPU, or after the execution of the previous instruction and the resolution of pertinent bus and interrupt requests. During the first microstate, the bus control signals issue BAR. During this state, the CPU will output the current contents of the PC onto the bus. BAR will strobe this data from the bus into the address register to fetch the instruction pointed to by the PC. Internally, the CPU increments PC and returns it to R7.

Beginning in the next micro state, the CPU will sample the BDRDY input line. If BDRDY is at a logic "1" level, the CPU will proceed to the next state where the fetched instruction will be inputted into the CPU via the signal DTB. If BDRDY is at a logic "0" level, the CPU will enter a WAIT state and remain in this state until BDRDY is brought to a logic "1" level. This WAIT state cannot be of indefinite duration; it must last no longer than 40 microseconds in order to preserve the dynamic characteristics of the CPU during this fetch sequence. The bus control signals will change to NACT throughout the WAIT state. External systems will need to utilize this WAIT state when their access times cannot meet the CPU speed requirements. As long as the CPU is in the WAIT state, it will sample the BDRDY line every microcycle (every 600 nanoseconds if the clock input is 3.3 MHz).

Once BDRDY is high, the CPU will issue the DTB bus signal. The external system presently addressed will gate data (in this instance the fetched instruction word) onto the bus. The instruction will be inputted to the CPU and loaded into its instruction register.

2.5.3 ADDRESS SEQUENCE

The Address Sequence will only be entered if the instruction just fetched is of an External Reference type. The distinctions between external and internal reference types will be made clearer in Section 3 "Instruction Set". Basically, External Reference instructions need to retrieve external data to complete execution of the instruction. For example, those instructions which specify the contents of a register to be the address of an operand are all of external reference type. Internal Reference instructions process operands already held within the registers of the CPU.

The Address Sequence is very similar to the Fetch Sequence. The CPU will begin by issuing BAR and outputting some computed effective address on the bus. The particular instruction will determine

how the effective address is computed. The effective address will be gated into the address register. The CPU will then sample BDRDY exactly as before. Again, if a WAIT state is entered, its duration must be less than 40 microseconds. Once BDRDY has been resolved, the CPU will issue a DTB which will gate the retrieved data onto the bus. The CPU inputs this data which will be used as an operand in executing the instruction.

NOTE: This sequence becomes slightly more complex if the instruction specifies direct addressing. The CPU will issue BAR, sample BDRDY, then issue ADAR to readdress memory, sample BDRDY again, and finally issue DTB.

2.5.4 EXECUTE SEQUENCE

After the data operands have been obtained, the CPU will execute the instruction. Execution can take as few as two microcode states or as many as six for the most complex instructions. Different types of bus control signals are activated according to instruction type. Examples are shown in the folllowing table.

Instruction Class	# of Micro States	Bus Control Signals
External Reference		
Input Group	2	DTB, NACT
Move Out	3	DW, DWS, then NACT
Double Byte Data	3	BAR, NACT, and then DTB
Internal Reference		
Register-Register	2	NACT
Shift	2 or 4	2 or 4 NACT

2.5.5 BUS REQUEST SEQUENCE

If there are no Bus Requests or Interrupt Requests following the execution of the present instruction, the CPU will return to the Fetch Sequence and fetch the next instruction from memory.

If some external device has activated the $\overline{\text{BUSRQ}}$ input line (by pulling it low) prior to the completion of the execution of the current interruptable instruction, the CPU will honor the Bus Request by causing the output signal $\overline{\text{BUSAK}}$ to go low. The CPU will enter a WAIT state and remain in this state until $\overline{\text{BUSRQ}}$ is brought to a logic "1" level. This WAIT state can be of indefinite duration. $\overline{\text{BUSAK}}$ will be low for as long as the CPU is in this state and the CPU will not use the bus; the bus control signals will be issuing NACT. All other signals will be in their inactive state.

The $\overline{\text{BUSRQ}}$ input signal can be activated asynchronously with respect to the CPU clocks. The CP1600 digitally synchronizes $\overline{\text{BUSRQ}}$ during every TS1 time.

2.5.6 INTERRUPT SEQUENCE

The CPU will enter its Interrupt Sequence after all bus requests have been resolved and either of the following conditions have been met:

a) The $\overline{\text{INTR}}$ input signal has been activated, i.e., it has been pulled to a logic "0" level in sufficient time for the CPU to recognize the input as a valid interrupt request; or,

b) The $\overline{\text{INTRM}}$ input signal has been similarly activated AND the internal interrupt flip-flop, INTFF, is set (enabled).

2.5 State Timing (cont.)

Once an interrupt request has been accepted by the CPU, the CPU performs the following functions automatically:

1. The first micro state will issue the bus control signal INTAK. The presence of this signal denotes acceptance by the CPU of the interrupt request and can be used by the peripheral devices to resolve interrupt priority. During this state, the CPU will output the current contents of Register 6, the Stack Pointer, to the Address Register in preparation for saving the program counter on the Stack.
2. The CPU will then output the PC as data to be stored and issue bus signals DW and DWS to write the Program Counter into the location pointed to by the Stack Pointer. That is, the CPU will store the PC into the top of the STACK. The Stack Pointer will be incremented after this store operation.
3. Finally, the CPU will issue bus signal IAB which will bring the starting address of the interrupt service routine into the program counter. The CPU will then return to the Fetch Sequence and execution continues in normal fashion from the new PC supplied by the interrupting device.

Again, in a manner similar to that of the $\overline{\text{BUSRQ}}$ line, the CPU samples the $\overline{\text{INTR}}$ and $\overline{\text{INTRM}}$ inputs every TS1 time. In addition, the CPU will honor Bus Requests before it will honor Interrupt Requests.

2.6 Timing Diagrams

The functional operation and relative timing of all CP1600 sequences are shown in the following diagrams. These include:

Initialization Sequence ..... Fig. 8
Instruction Fetch ..... Fig. 9
Instruction Fetch & Data Access .....Figs. 10a, 10b
Write Sequence ..... Fig. 11
Branch Sequence .....Fig. 12a
External Condition Test Sequence .....Fig. 12b
TCI Instruction Timing ..... Fig. 13
$\overline{\text{PCIT}}$ Timing ..... Fig. 14
Interrupt Sequence ..... Fig. 15
$\overline{\text{BUSRQ}}$/$\overline{\text{BUSAK}}$ Sequence..... Fig. 16
BDRDY Timing ..... Fig. 17

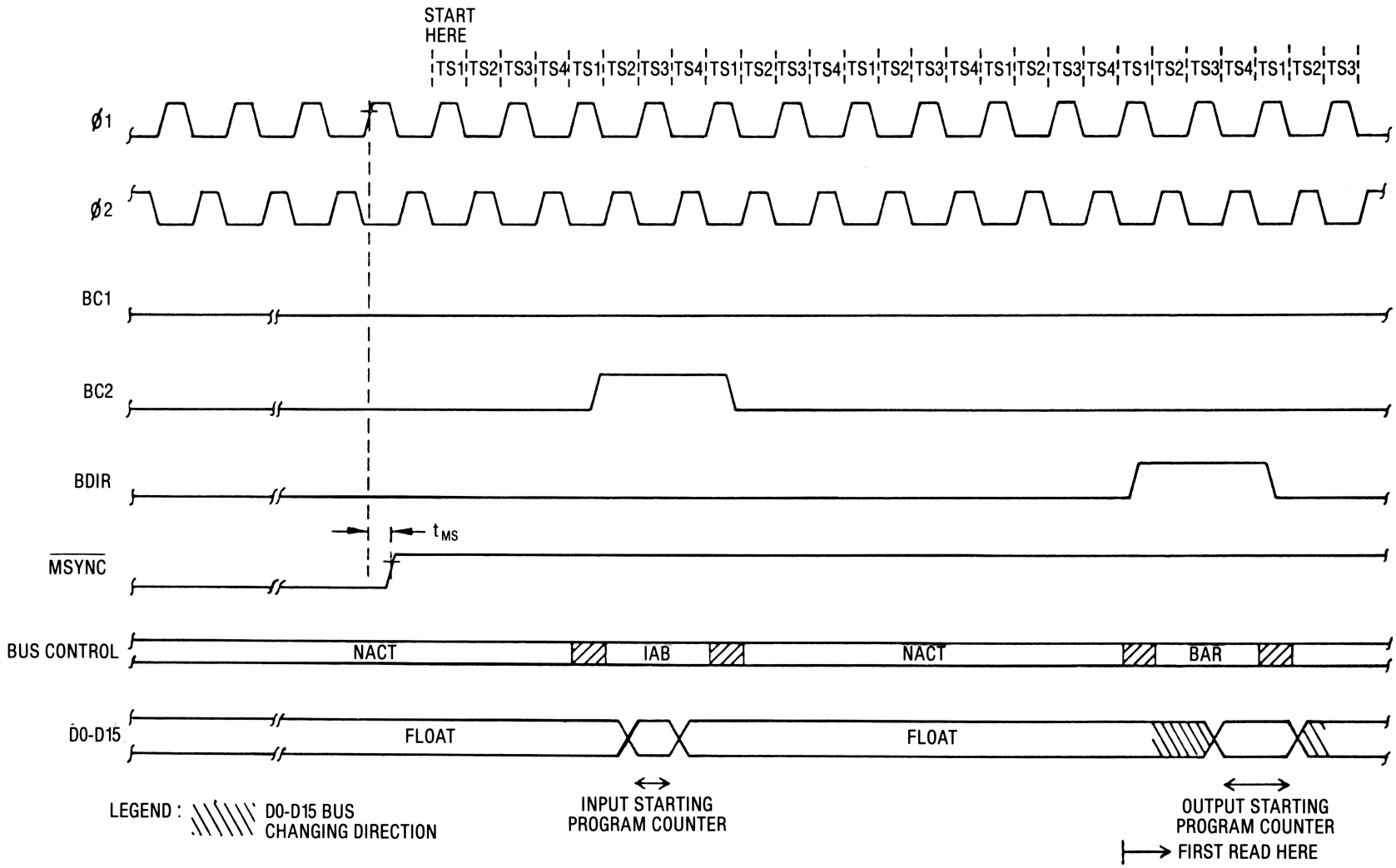
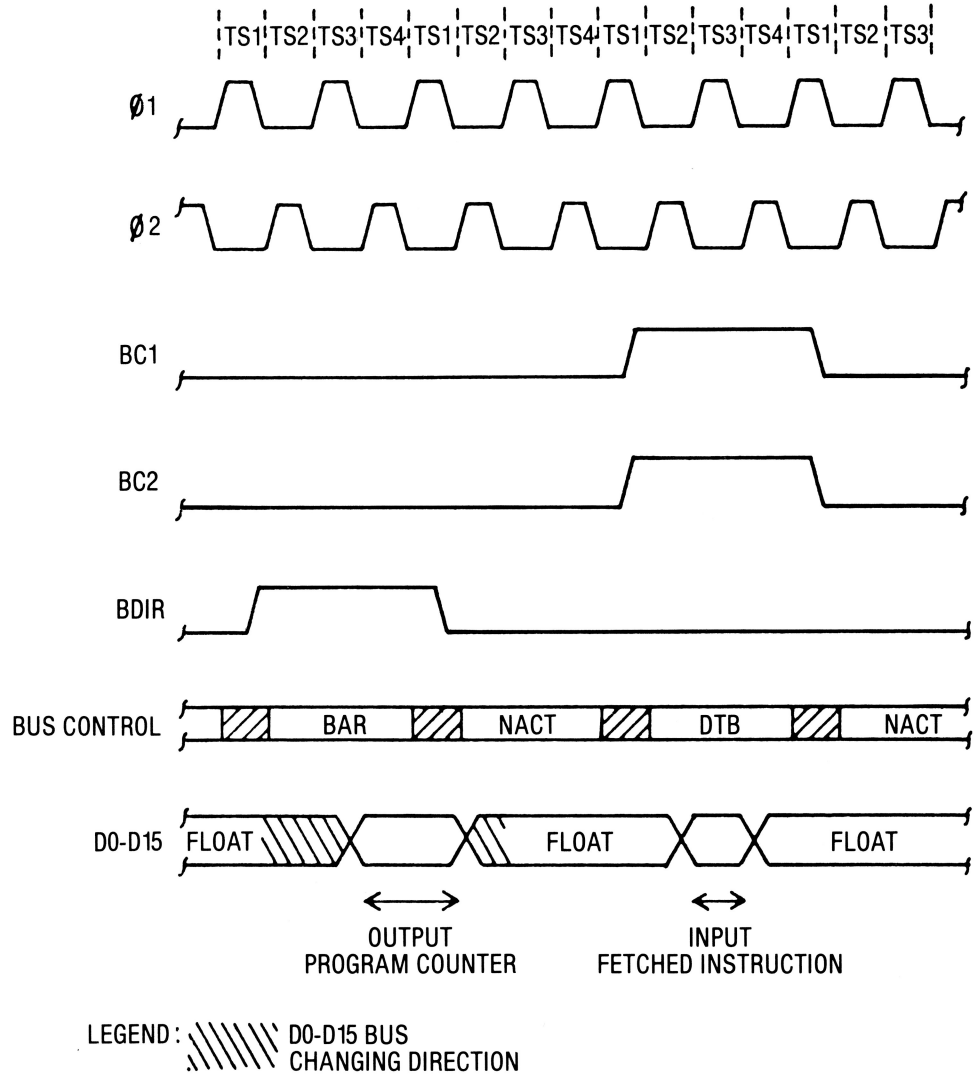


Fig. 8 INITIALIZATION SEQUENCE

Fig. 9 INSTRUCTION FETCH



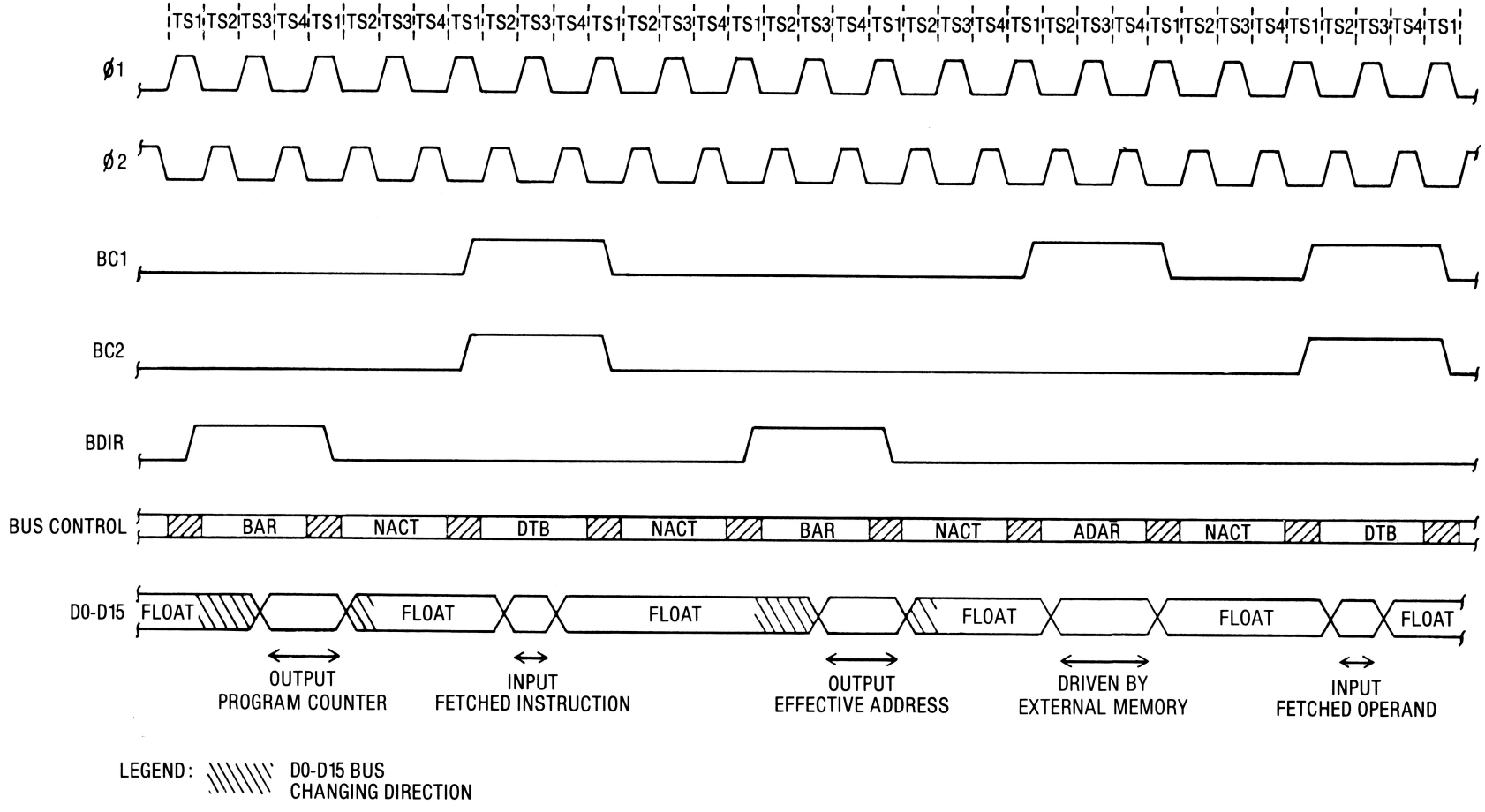
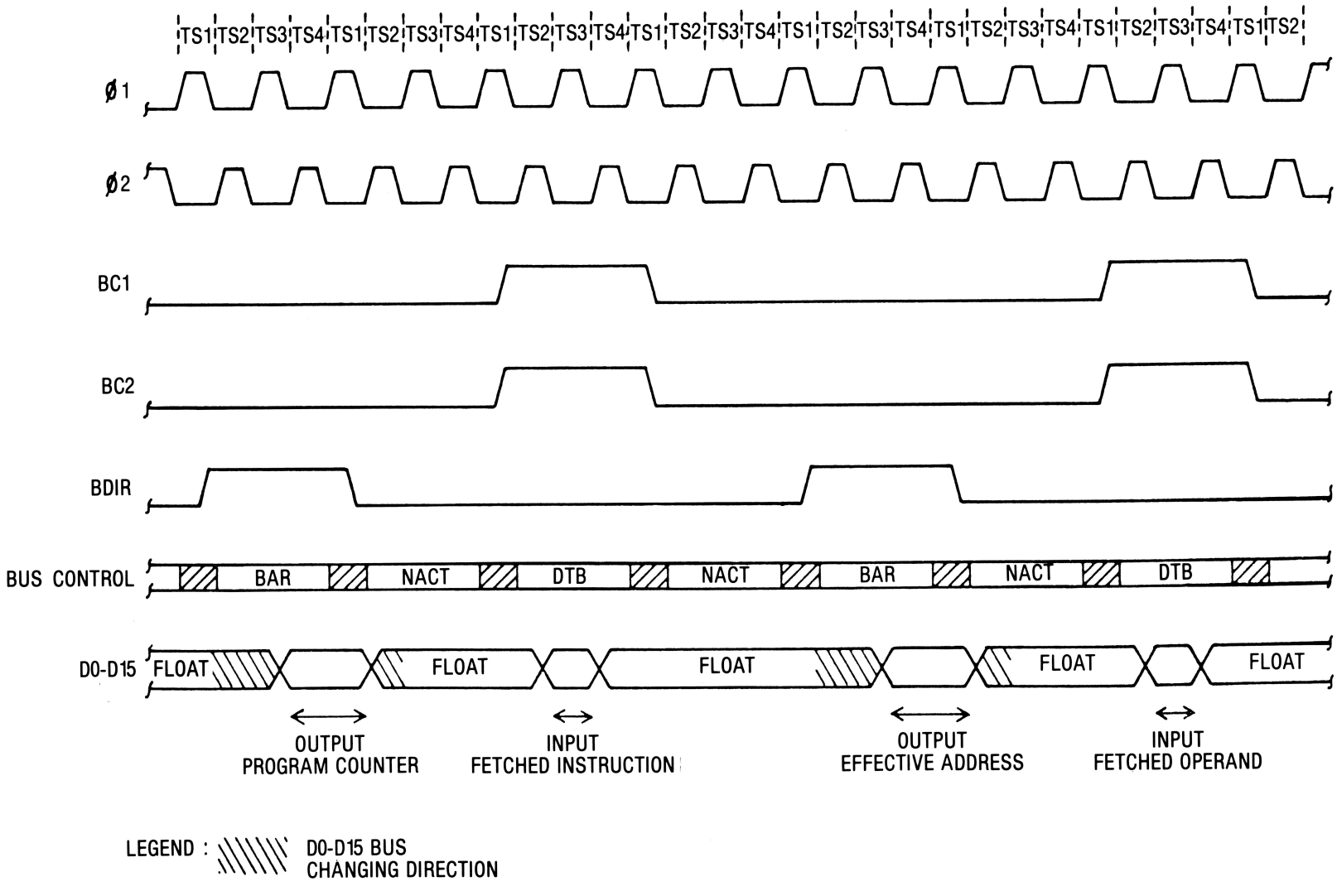


Fig. 10a INSTRUCTION FETCH AND DATA ACCESS (Direct address mode)

Fig. 10b INSTRUCTION FETCH AND DATA ACCESS (Indirect address mode)



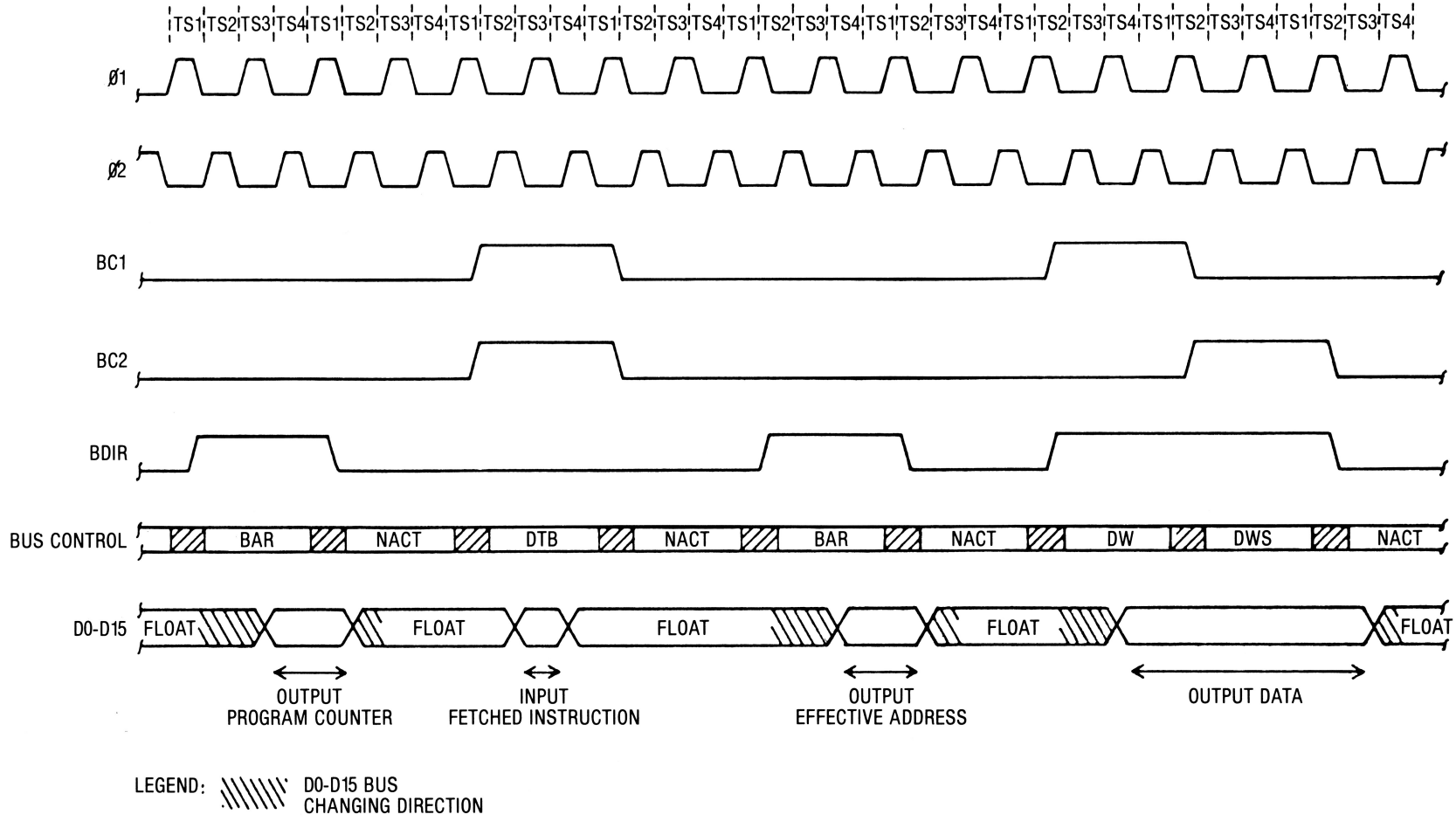
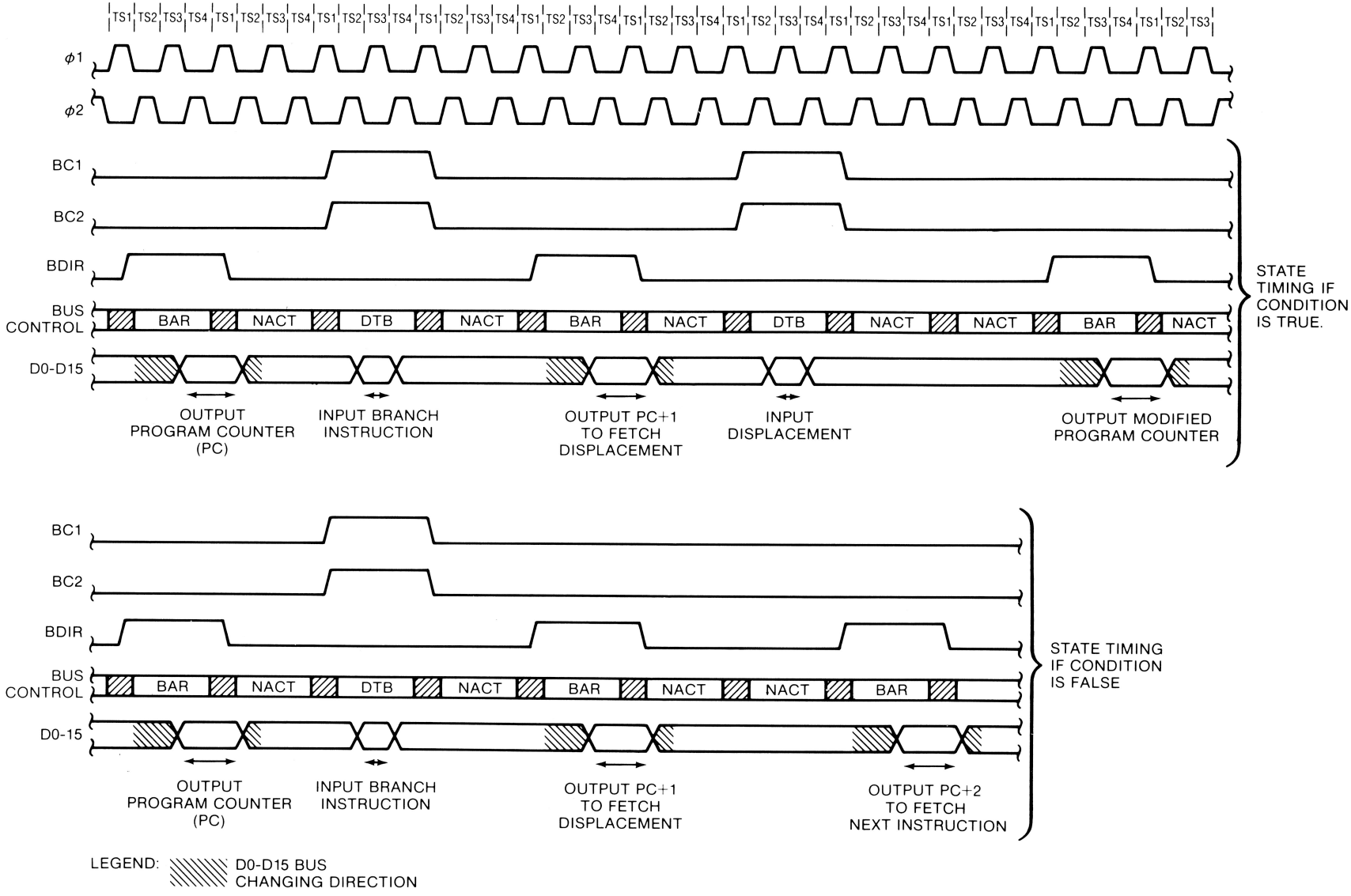


Fig. 11 WRITE SEQUENCE (MVO timing-indirect addressing)

Fig. 12a BRANCH SEQUENCE


TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3
φ1
φ2
BC1
BC2
BDIR
BUS CONTROL
BAR NACT DTB NACT BAR NACT DTB NACT NACT BAR NACT
D0-D15
STATE TIMING IF CONDITION IS TRUE.
OUTPUT PROGRAM COUNTER (PC)
INPUT BRANCH INSTRUCTION
OUTPUT PC+1 TO FETCH DISPLACEMENT
INPUT DISPLACEMENT
OUTPUT MODIFIED PROGRAM COUNTER
BC1
BC2
BDIR
BUS CONTROL
BAR NACT DTB NACT BAR NACT NACT BAR
D0-15
STATE TIMING IF CONDITION IS FALSE
OUTPUT PROGRAM COUNTER (PC)
INPUT BRANCH INSTRUCTION
OUTPUT PC+1 TO FETCH DISPLACEMENT
OUTPUT PC+2 TO FETCH NEXT INSTRUCTION
LEGEND: D0-D15 BUS CHANGING DIRECTION

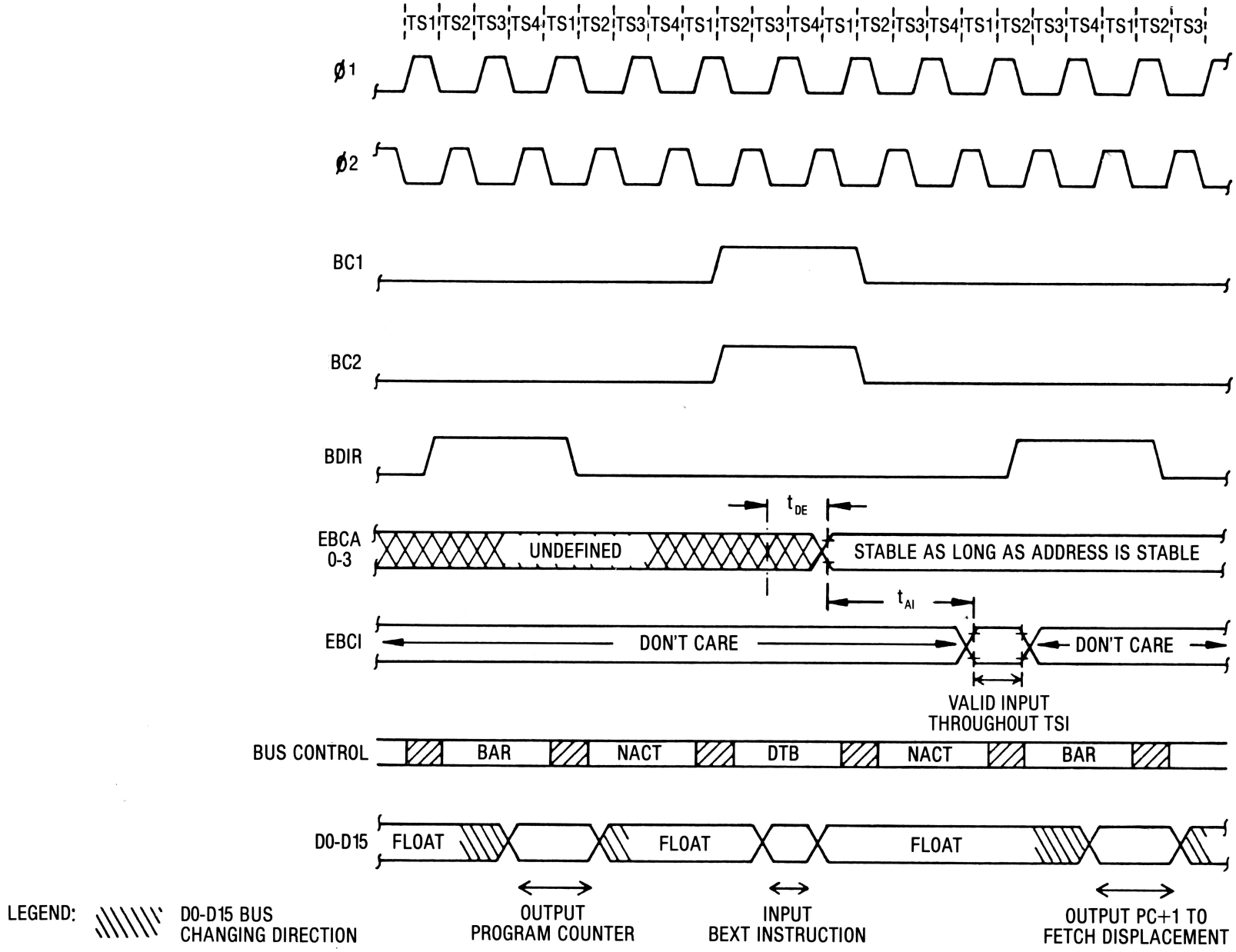
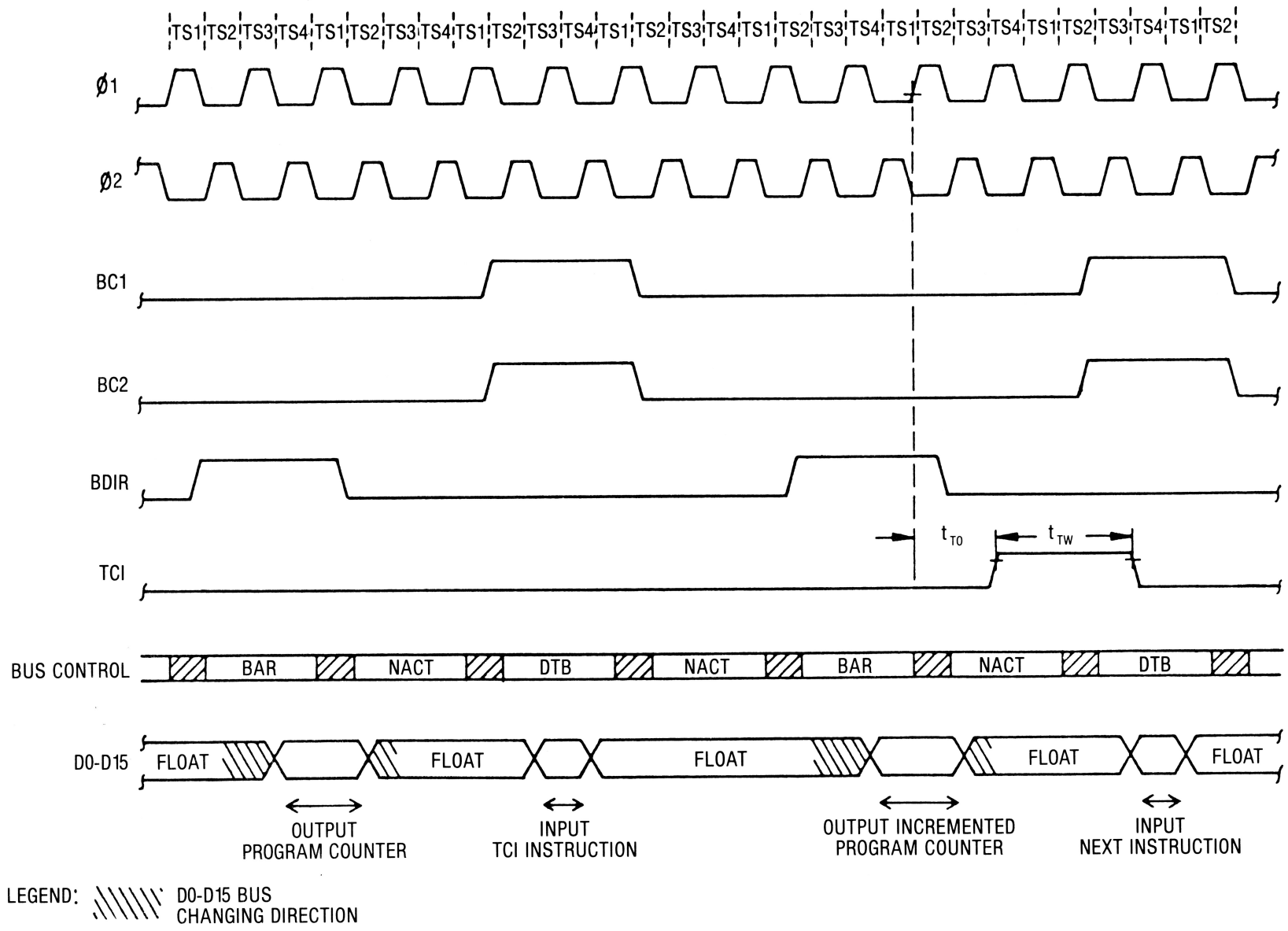


Fig. 12b EXTERNAL CONDITION TEST SEQUENCE (Branch on external instruction)

Fig. 13 TCI INSTRUCTION TIMING



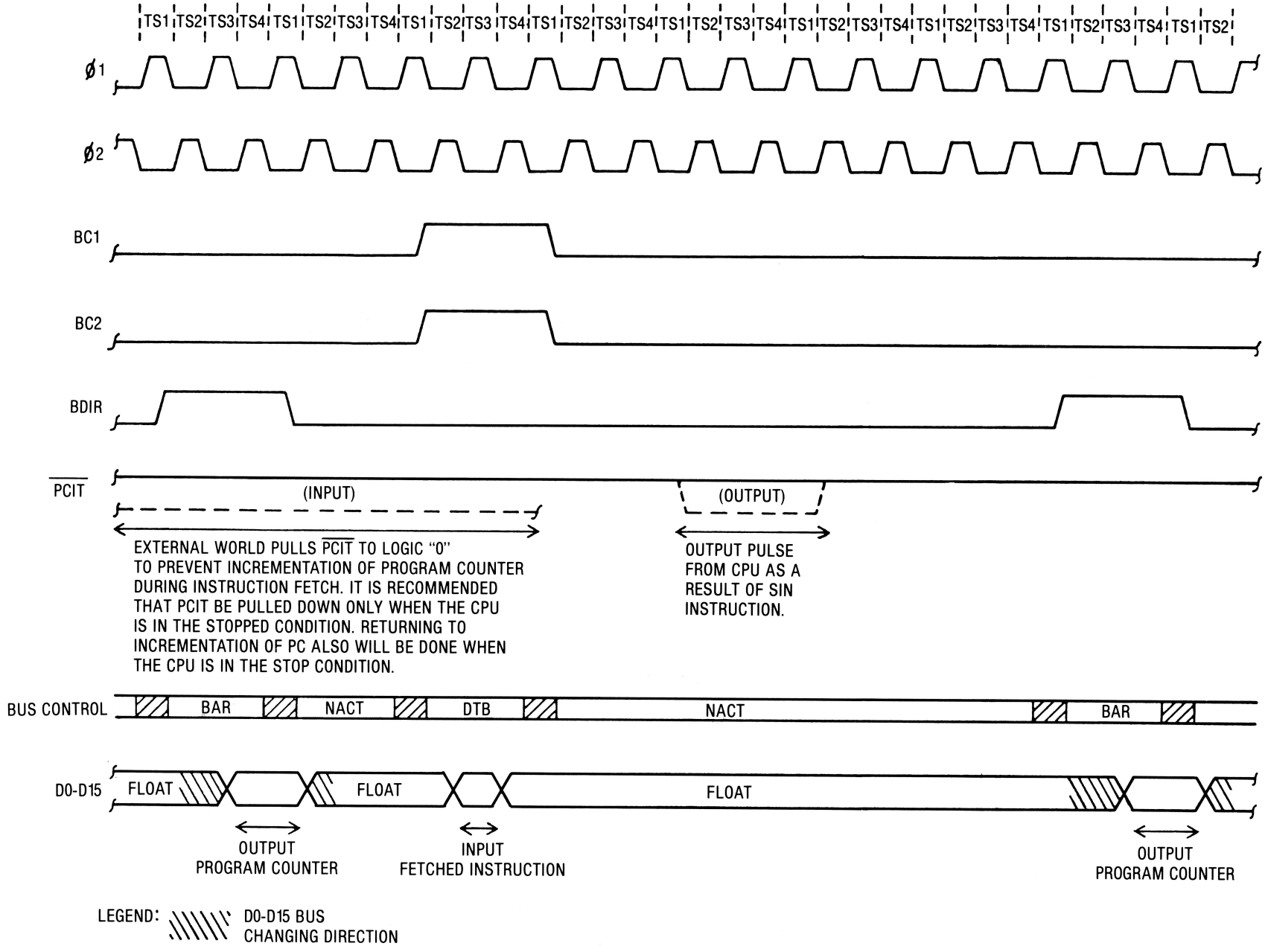
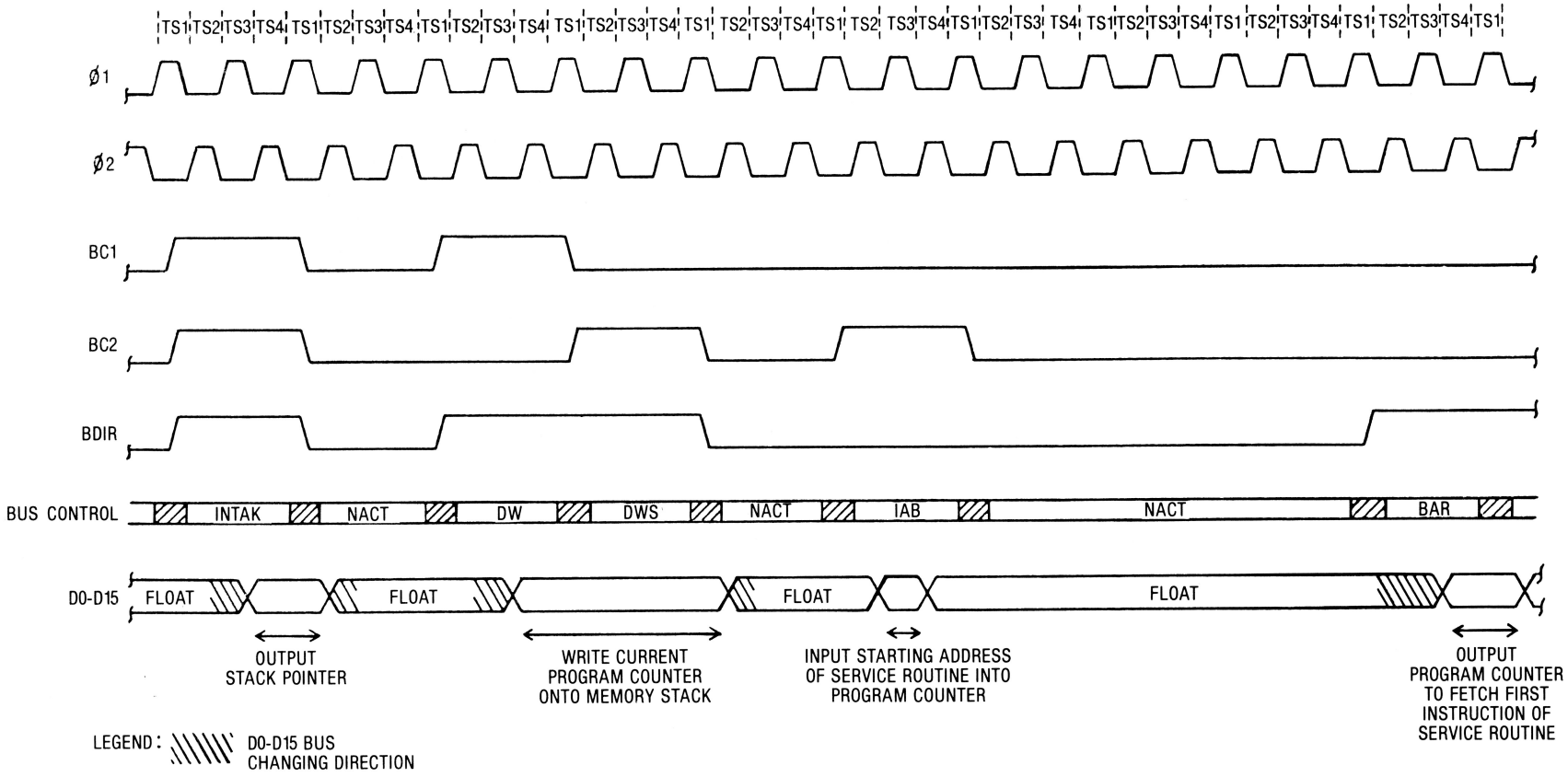

TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2 TS3 TS4 TS1 TS2
Ø1
Ø2
BC1
BC2
BDIR
PCIT
(INPUT)
(OUTPUT)
EXTERNAL WORLD PULLS PCIT TO LOGIC "0" TO PREVENT INCREMENTATION OF PROGRAM COUNTER DURING INSTRUCTION FETCH. IT IS RECOMMENDED THAT PCIT BE PULLED DOWN ONLY WHEN THE CPU IS IN THE STOPPED CONDITION. RETURNING TO INCREMENTATION OF PC ALSO WILL BE DONE WHEN THE CPU IS IN THE STOP CONDITION.
OUTPUT PULSE FROM CPU AS A RESULT OF SIN INSTRUCTION.
BUS CONTROL
BAR
NACT
DTB
NACT
BAR
D0-D15
FLOAT
FLOAT
FLOAT
OUTPUT PROGRAM COUNTER
INPUT FETCHED INSTRUCTION
OUTPUT PROGRAM COUNTER
LEGEND:
D0-D15 BUS CHANGING DIRECTION

Fig. 14 PCIT TIMING

Fig. 15 INTERRUPT SEQUENCE



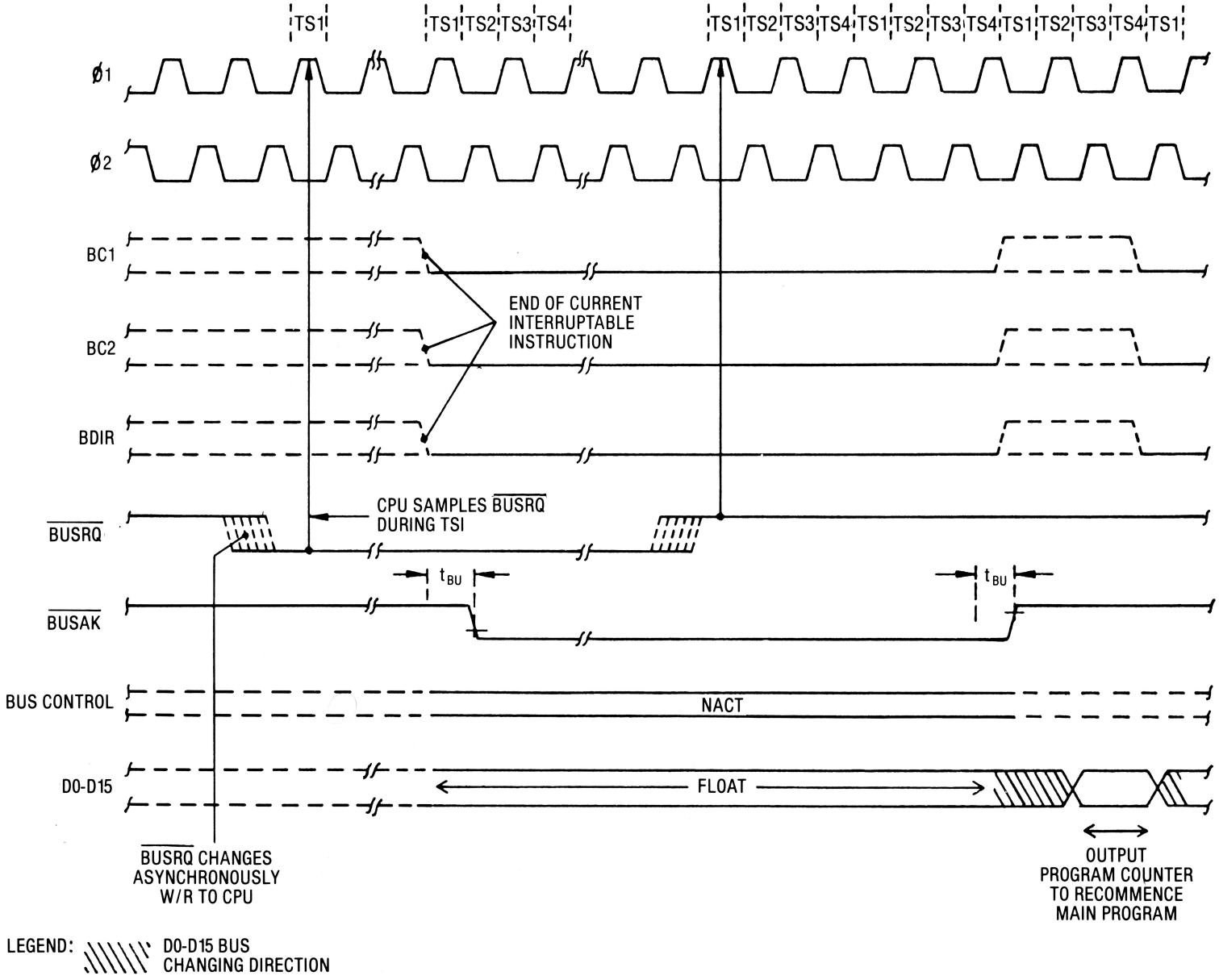
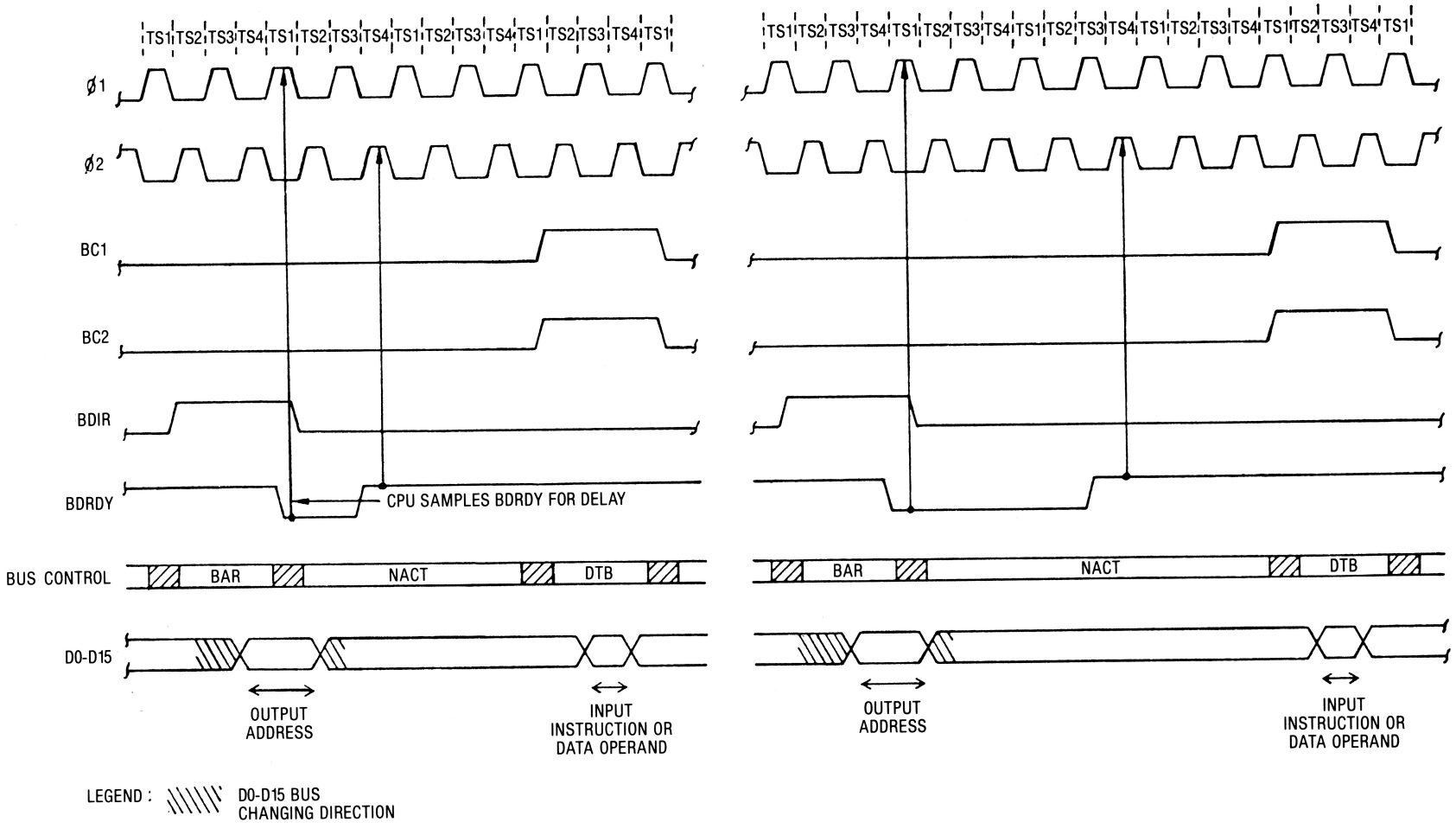


Fig. 16 BUS REQUEST (BUSRQ)/BUS ACKNOWLEDGE (BUSAK) SEQUENCE

Fig. 17 BUS DATA READY (BDRDY) TIMING



3 INSTRUCTION SET

3.1 General Instruction Format

The basic instruction word format for the CP1600 microprocessor consists of 10 bits located in the low order bit positions of a 16-bit processor word. The high order 6 bits of every 16-bit word supplied to the CP1600 as an instruction word are ignored by the internal microcontrol logic. This results in a compact, highly efficient 10-bit instruction word that can be implemented directly in a single 10-bit wide ROM instead of dual 8-bit wide ROMs for those systems requiring the ultimate in ROM bit efficiency. The 8 internal working registers, all data operands, displacement constants, and arithmetic and logical operations are implemented to full 16-bit significance providing the performance, flexibility, and efficiency of 16-bit operation.

The 10-bit instruction word is divided into 3 functional fields to which most instructions conform.

15 - - - - 10	9	8 7 6	5 4 3	2 1 0
Don't Care*	OP Code		F1	F2

*This field is not implemented by the microcontrol logic in the CP1600.

The most significant 4 bits of the CP1600 instruction word is the Operational Field which provides 16 basic operations. The 16 operations are evenly divided into 2 groups of 8 operations each: the External Reference group and the Internal Reference group.

3.1.1 INSTRUCTION OVERVIEW

All Series 1600 symbolic instructions are expressed as statements containing an operator (OP Code) and in most cases one or two operands (Source/Address Mode and Destination). In general, the first three characters of an operator mnemonic represent the operating mode, such as ADD@ for ADD indirect. The following description of instructions depicts both the symbolic format and also the binary translation. The timing, status register change, and instruction description is also shown.

3.1 General Instruction Format (cont.)

3.1.2 SYMBOLIC NOTATION

Address Modes:	R	- register
	'blank'	- direct address
	I	- immediate data
	@	- indirect address
Functions:	+	- addition
	−	- subtraction
	V	- inclusive OR
	∀	- exclusive OR
	Λ	- AND
	( )	- contents of
	←	- is replaced by
	<>	- optional operand
Status:	S	- Sign bit
	Z	- Zero bit
	C	- Carry bit
	OV	- OVerflow bit
Operands:	SSS	- Source Register
	DDD	- Destination Register
	MMM	- Register Address Mode
	RR	- Register (0-3)
	n	- Number of Shifts
	S	- Sign of Address Displacement
	EEEE	- External Condition Code (0-15)
	DA	- Destination Address
	SA	- Source Address
	I-I	- Immediate data word
	P-P	- Address displacement for Branch

3.1.3 OPERATION OF STATUS BITS

During arithmetic and logical operations in the CPU, the ALU Status Bits are used to monitor and record four characteristics of the resultant data. These bits are Carry (C) out of the ALU, arithmetic OVerflow (OV) from the ALU, Zero (Z) result from the output of the Shifter, and Sign (S) detect from the output of the Shifter. These bits operate in the following manner:

C - set if the result of an ADD, SUB, or CMP instruction produces a carry; cleared otherwise (SUB and CMP are performed by addition of two's complement of the subtrahend operand).

OV - set if the result of an ADD, SUB, or CMP instruction produces arithmetic overflow (two's complement notation); cleared otherwise (SUB and CMP are performed by addition of two's complement of the subtrahend operand). Logically defined as the exclusive OR of the carry out of the two most significant bits of the arithmetic operation.

Z - set if the result of an operation produces an all zero result; cleared otherwise.

S - set if the result of an operation contains a "one" in the high order bit position (bit 15); cleared otherwise. Any value with bit 15 of a register set will be considered negative in arithmetic operations.

3.2 Internal Reference Instructions

The Internal Reference instructions, which have a zero in the high order bit position of the Operation Field, process data already held within the 8 working registers of the CP1600. These instructions are grouped into register to register, single register, register shift, internal control, and jump instructions. The basic formats are depicted in the following paragraphs.

3.2.1 REGISTER TO REGISTER

15 - - - - - 10	9	8 7 6	5 4 3	2 1 0
Don't Care	0	OP Code	S S S	D D D

OP CODE	OPERATION	
000	(See 3.2.2, Single Register Operations)	
001	(See 3.2.3, Register Shift Operations)	
010	MOVe Register	(MOVR)
011	ADD Register	(ADDR)
100	SUBtract Register	(SUBR)
101	CoMPare Register	(CMPR)
110	logical AND Register	(ANDR)
111	eXclusive OR Register	(XORR)

3.2.2 SINGLE REGISTER

15 - - - - - 10	9	8 7 6	5 4 3	2 1 0
Don't Care	0	0 0 0	OP Code	D D D

OP CODE	OPERATION	
000	(See 3.2.4, Internal Control Operations)	
001	INCrement Register	(INCR)
010	DECrement Register	(DECR)
011	COMplement Register	(COMR)
100	NEGate (2's complement) Register	(NEGR)
101	ADd Carry bit Register	(ADCR)
110	*Get Status WorD register	(GSWD)
111	Restore Status WorD register	(RSWD)

*The GSWD instruction can use only R0, R1, R2, or R3 as register specifications for true Get Status Word operation. Use of R4 or R5 results in a NOP instruction. Use of R6 or R7 results in a NOP internal to the CP1600 and causes a pulse to be outputted on the $\overline{\text{PCIT}}$ pin. This can be used to cause an interrupt to form a Software Interrupt (SIN) or Trap instruction.

3.2.3 REGISTER SHIFT

15 - - - - - 10	9	8 7 6	5 4 3	2 1 0
Don't Care	0	0 0 1	OP Code	N D D

OP CODE	OPERATION	
000	SWAP 8-bit bytes	(SWAP)
001	Shift Logical Left	(SLL)
010	Rotate Left thru Carry	(RLC)
011	Shift Logical Left thru Carry	(SLLC)
100	Shift Logical Right	(SLR)
101	Shift Arithmetic Right	(SAR)
110	Rotate Right thru Carry	(RRC)
111	Shift Arithmetic Right thru Carry	(SARC)

N=0: 1 position shift
N=1: 2 position shift

NOTE: Shifts can only be performed on Registers R0-R3.

3.2 Internal Reference Instructions (cont.)

3.2.4 INTERNAL CONTROL

15 - - - - - 10	9	8	7	6	5	4	3	2	1	0
Don't Care	0	0	0	0	0	0	0	OP Code		

OP CODE	OPERATION	
000	*HaLT	(HLT)
001	Set Double Byte Data	(SDBD)
010	Enable Interrupt System	(EIS)
011	Disable Interrupt System	(DIS)
100	(See 3.2.5, Jump/Jump and Save Return Operations)	
101	Terminate Current Interrupt	(TCI)
110	CLeaR Carry Bit	(CLRC)
111	SET Carry Bit	(SETC)

* Halt after next interruptable instruction is executed.

3.2.5 JUMP/JUMP AND SAVE RETURN

	15 - - - - - 10	9	8	7	6	5	4	3	2	1	0
PC	Don't Care	0	0	0	0	0	0	0	1	0	0
PC+1	Don't Care	B	B	A	A	A	A	A	A	D	E
PC+2	Don't Care	A	A	A	A	A	A	A	A	A	A
PC+3											

BB		DE		OPERATION
11	Jump	00	—	(J)
		01	Enable interrupt	(JE)
		10	Disable interrupt	(JD)
≠11	Jump and Save Return	00	—	(JSR)
		01	Enable interrupt	(JSRE)
		10	Disable interrupt	(JSRD)

The instructions in the jump group occupy three consecutive memory locations and are used to perform direct jumps and subroutine calls. In addition the interrupt system can be either enabled or disabled by bits 0-1 of the second word of the instruction.

On direct jumps the return address is not saved. The destination address is formed by using bits 2-7 of the second word as bits 10-15 of the jump address and using bits 0-9 of the third word as bits 0-9 of the jump address.

On subroutine jumps, the return address (PC+3) is saved in register 4, 5, or 6. This register designation is formed by using bits 8-9 of the second word (field BB) and adding 4. Note if BB=3, the designated register becomes 7 (the PC) which is the jump instruction. By using one of the Jump and Save instructions, the linkage to a subroutine is accomplished with the return address saved in register 1BB. This is a very efficient means of obtaining subroutine linkage (see 3.4.8 for examples).

3.3 External Reference Instructions

The External Reference instructions, which have a one in the high order bit position of the Operation Field, route data into or out of the CP1600 to externally located bus addresses. Since a single address bus structure is utilized in the CP1600 architecture, both memory and peripheral devices reside in the same Address Space. Only the system address allocation, as defined by the user, differentiates memory from I/O devices; therefore, no special input/output instructions are required and any External Reference instruction can access memory or peripheral devices. The CP1600 utilizes a full 16-bit addressing structure which provides access to 65,536 unique locations.

3.3.1 DATA ACCESS INSTRUCTIONS

The format for 7 of the 8 External Reference instructions is shown below.

15 - - - - 10	9	8 7 6	5 4 3	2 1 0
Don't Care	1	OP Code	Mode	D D D

OP CODE	OPERATION	
001	MoVe Out	(MVO)
010	MoVe In	(MVI)
011	ADD	(ADD)
100	SUBtract	(SUB)
101	CoMPare	(CMP)
110	logical AND	(AND)
111	eXclusive OR	(XOR)

MODE		
000	—	Direct address in next memory location
001	—	Indirect address thru R1
010	—	Indirect address thru R2
011	—	Indirect address thru R3
100	—	Indirect address thru R4, R4+1→R4
101	—	Indirect address thru R5, R5+1→R5
110	—	Stack thru R6; Post Increment if Output; Pre-Decrement if Input
111	—	Immediate data in next memory location

Any of the 7 operations can be used with any of the 8 modes.

These instructions use the lower 3 bits of the word as the register designator. This register is the source of data for output operations and the destination for data on input operations. The Mode Field of the instruction specifies the addressing mode to be used to access the Address Space. Mode 0 indicates that the contents of the next memory location is to be used directly as an external address. Modes 1, 2 and 3 indicate that registers 1, 2 and 3 respectively contain the external address. Modes 4 and 5 indicate indirect addressing through registers 4 and 5 with the contents of the register automatically incremented after use as the address; i.e., post increment. Mode 6 indicates stack operations, i.e., indirect through register 6 with the contents of the register post incremented for output operations (push) and pre-decremented for input operations (pull). The Stack is maintained by register 6 and expands upward in external memory from low address to high address. Mode 7 indicates that the next memory location holds data to be used directly; i.e., immediate data.

3.3 External Reference Instructions (cont.)

3.3.2 CONDITIONAL BRANCH INSTRUCTIONS

The eighth instruction in the External Reference group is the Conditional Branch type which has the format:

	15 - - - - 10	9	8	7	6	5	4	3	2	1	0
PC	Don't Care	1	0	0	0	±	I/E	Condition Code			
PC+1	P P P P P P	P	P	P	P	P	P	P	P	P	P

This instruction tests any one of 32 conditions and causes a branch in the program if the test has a true result. The 32 conditions are evenly divided into 2 groups of 16 each. The Internal Condition Codes (Bit 4=0) comprise various tests on the state of the status register bits of the ALU while the External Condition Codes (Bit 4=1) comprise 16 tests on external digital status signals of the user's choice.

In case of a false result, the program continues to the next sequential instruction. When the appropriate conditions are true, the instruction transfers program control to the specified destination. The actual destination address is determined by adding or subtracting the address displacement contained in the second word of the instruction (P-P) to the program counter. If the destination direction bit of the instruction (±) is zero, then the displacement is added to PC+2. If it is one, then the displacement is one's complemented and then added to PC+2.

3.3.3 EXTERNAL REFERENCE ADDRESSING MODES

All of the instructions in the external reference group require that data be accessed from the external Address Space of the processor. The detailed sequence for each of the various addressing modes follows.

The effective address (EA) is used to access the first operand. The second operand is always contained in an internal CPU register with the result of the operation replaced in this same register. Note that the program counter (R7) is incremented before the Operation Code is performed; thus, if the destination register (DDD) specified by the instruction is R7, the incremented program counter is modified creating the effect of a branch in the program sequence.

Direct Address (Mode=000)

	Don't Care	1	OP Code	0	0	0	D	D	D
PC									
PC+1	A A A A A A A A A A A A A A A A								

PC←PC+2

EA←AAAAAAAAAAAAAAAA

Immediate Data (Mode=111)

	Don't Care	1	OP Code	1	1	1	D	D	D
PC									
PC+1	I I I I I I I I I I I I I I I I								

PC←PC+2

EA←PC+1

Indirect Address (Mode=001 to 110)

PC	Don't Care		1	OP Code			M	M	M	D D D

MMM=001, 010, 011

PC←PC+1
EA←(MMM)

MMM=100, 101

EA←(MMM)
(MMM)←(MMM)+1

MMM=110 for MVO only

PC←PC+1
EA←(MMM)
(MMM)←(MMM)+1

MMM=110 for non-MVO

PC←PC+1
(MMM)←(MMM)-1
EA←(MMM)

3.3.4 DOUBLE BYTE DATA

In order to utilize an efficient 10-bit instruction ROM and also reference 16-bit data from the same memory area, a Double Byte Data feature is available on the CP1600. When a Set Double Byte Data (SDBD) instruction precedes an external reference instruction other than MVO, two words of memory are accessed. The least significant data byte is obtained from the low order eight bits of the first word and the most significant data byte is obtained from the low order eight bits of the second word. These bytes are then combined by the processor to form a 16-bit data word. Double byte data may be accessed via indirect addressing modes 4, 5 and 7. If modes 1, 2 and 3 are used only one word is accessed with both bytes of data obtained from the lower eight bits of the same word. Modes 0 and 6 are not supported in the double byte data operation.

Immediate Double Byte Data

PC-1	Don't Care		0	0	0	0	0	0	0	0	1 *
PC	Don't Care		1	OP Code			1	1	1	D	D D
PC+1	Don't Care					L	L	L	L	L	L L L
PC+2	Don't Care					U	U	U	U	U	U U U

* SDBD instruction

PC←PC+3
EA_L←(PC+1)
EA_U←(PC+2)

NOTE: The assembler will automatically generate the SDBD instruction preceding the external reference instruction when the constant specified by the assembly language statement exceeds the memory word size specified. If the user specifies the SDBD, then the data word will always be generated to fill two bytes.

Indirect Addressed Double Byte Data

PC−1	Don't Care		0	0	0	0	0	0	0	0	1 *
PC	Don't Care		1	OP Code			M	M	M	D	D D

* SDBD instruction

PC←PC+1

If MMM=1, 2, 3
EA_L←(MMM)
EA_U←(MMM)

If MMM=4, 5
EA_L←(MMM)
(MMM)←(MMM)+1
EA_U←(MMM)
(MMM)←(MMM)+1

3.4 Program Examples

The following examples demonstrate common program coding techniques for use on the CP1600. These include loop control, array addressing, table look-up, use of the stack, reentrant interrupt service, subroutine calls, and other miscellaneous ideas. All are intended to be useful examples but in many cases are designed to show the use of particular instructions as opposed to minimizing code or optimizing speed of execution.

3.4.1 LOOP OR ITERATION CONTROL

Counting down using a general register:

```
        MVII    n, R3           ; set loop count
LOOP    .
        .
        .
        .
        DECR    R3              ; decrement count
        BNZE    LOOP            ; if not zero, continue loop
                                ; loop complete
```

Counting up using a general register:

```
        CLRR    R2              ; zero count
LOOP    .
        .
        .
        .
        .
        .
        INCR    R2              ; increment count
        CMPI    n, R2           ; compare to limit
        BNEQ    LOOP            ; if not equal, continue loop
                                ; loop complete
```

Counting down using a storage cell:

```
        MVII    n, R0           ; set loop count
LOOP    MVO     R0, CNTR        ; store count
        .
        .
        .
        .
        .
        .
        MVI     CNTR, R0        ; fetch count
        DECR    R0              ; decrement count
        BNZE    LOOP            ; if not zero, continue loop
                                ; loop complete
```

Loop control by address limiter:

```
        MVII    ADDR, R5        ; set starting address
LOOP    ADD@    R5, R0          ; fetch data, increment pointer
          .
          .
          .
          .
          .
          .
          .
        CMPI    ADDR+n, R5 ; compare for final address
        BNEQ    LOOP            ; if not equal, continue loop
                                ; loop complete
```

3.4.2 TABLE ACCESS

Sequential ascending access:

```
        MVII    n, R3           ; set count
        MVII    TBL, R4         ; set table base address
LOOP    MVI@    R4, R0          ; fetch table item & increment
                                  address
          .
          .
          .
        DECR    R3              ; decrement count
        BNZE    LOOP            ; continue until all items accessed
          .                     ; complete
          .
          .
```

Sequential descending access:

```
        MVII    n, R3           ; set count
        MVII    TBL+n, R2       ; set table end address
LOOP    MVI@    R2, R0          ; fetch table item
        DECR    R2              ; decrement table address
          .
          .
          .
        DECR    R3              ; decrement count
        BNZE    LOOP            ; continue until all items accessed
          .                     ; complete
          .
          .
```

Random access; ie, TBL (I):

```
          .
          .
          .
          .
        MVI     I, R1           ; fetch index
        ADDI    TBL-1, R1       ; compute table pointer
        MVI@    R1, R0          ; fetch table item
          .
          .
          .
```

3.4 Program Examples (cont.)

3.4.3 TRANSFER VIA AN INDEXED DISPATCH TABLE (1≤I≤4)

GO TO (a, b, c, d) I

```
        MVI     I, R1       ; fetch dispatch index
        ADDI    TBL-1, R1   ; compute pointer to table
        MVI@    R1, R7      ; move dest addr to prog counter
TBL     WORD    a, b, c, d  ; destination addresses
```

3.4.4 EVALUATE VARIABLES AND TRANSFER

IF (I. LT. J) GO TO a

```
MVI     I, RO       ; fetch I variable
CMP     J, RO       ; compare I to J
BLT     a           ; branch if less than
```

3.4.5 ARITHMETIC OPERATIONS

Add variables; ie, I=J+K+L:

```
MVI     J, R0       ; fetch J
ADD     K, R0       ; add K
ADD     L, R0       ; add L
MVO     R0, I       ; move sum to I
```

Add and subtract variables; ie, I=(J+K)−(L-M):

```
MVI     J, R0       ; fetch J
ADD     K, R0       ; add K
MVI     L, R1       ; fetch L
SUB     M, R1       ; subt. M
SUBR    R1, R0      ; subtract L-M from J+K
MVO     R0, I       ; move result to I
```

Double precision register to register add:

```
ADDR    R3, R1      ; add lower words
ADCR    R0          ; add in any carry
ADDR    R2, R0      ; add upper words
```

Double precision memory to register subtract:

```
DECR    R0          ; correct upper for add carry
SUB@    R4, R1      ; subtract lower halves, incr. pntr.
ADCR    R0          ; add in any carry
SUB@    R4, R0      ; subtract upper halves
```

3.4.6 STACK OPERATIONS

When register R6 is used to address storage indirectly, post-incrementing and pre-decrementing are performed by the CP1600 automatically. When data is moved out of the CPU via R6, it is placed in the storage cell specified by the contents of the register and then the contents of R6 is incremented by one. When data is moved into the CPU via R6, the contents of R6 is first decremented then data is fetched from the storage cell specified by R6. Thus, R6 can be used to manage a dynamic last in, first out storage area commonly called a stack.

The stack is used by the CP1600 as a temporary storage area to hold interrupt return addresses. It can also be used to hold nested subroutine return addresses, data during general programming activities, and operands to be evaluated using polish notation techniques. For user convenience, the Series 1600 assemblers recognize the mnemonics PSHR r and PULR r as the equivalent of MVO @ r, R6 and MVI @ R6, r respectively. PSHR r causes the contents of register r to be pushed onto the top of the stack maintained by R6 (higher in memory) and conversely PULR r pulls the contents of the top of the stack into register r (lower in memory).

Save registers:

```
        PSHR    Rn
```

Restore registers:

```
        PULR    Rn
```

Add top two elements in stack and compare to third element:

```
        PULR    R0          ; fetch top element
        ADD@    R6, R0      ; add second element
        CMP@    R6, R0      ; compare to third
        BEQ     MATCH       ; if equal, go to MATCH
```

3.4.7 INTERRUPT PROCESSING

Reentrant interrupt processing; i.e., recognizing and processing higher priority interrupts while currently servicing a lower priority one, requires totally reentrant CPU context save and restore routines.

Context save:

```
CNXSAV  PSHR    R0          ; save R0
        GSWD    R0          ; fetch CPU status
        PSHR    R0          ; save status
        PSHR    R1          ; save all registers
        PSHR    R2          ; required for
                            ; interrupt service
                            ; routine up
        PSHR    Rn          ; to Rn
```

Note that the address of the instruction following an interrupted instruction is pushed onto the stack by the interrupt process.

Context restore:

```
CNXRST  PULR    Rn          ; restore Rn
          .
          .
          .
          .
        PULR    R2          ; restore R2
        PULR    R1          ; restore R1
        PULR    R0          ; get status
        RSWD    R0          ; replace status
        PULR    R0          ; restore R0
        PULR    R7          ; return to interrupted process
```

3.4 Program Examples (cont.)

3.4.8 SUBROUTINES

A subroutine is a common sequence of instructions which can be executed many times by being called from many different places in a program. The JSR (jump and save return) instruction provides for entry into subroutines while saving the return address in register 4, 5, or 6. Data passed to a subroutine by the calling program (arguments) may be transferred by values or addresses held in registers or as items stored sequentially following the JSR instruction.

Subroutine call with no arguments:

```
        JSR     R5, SUBR    ; transfer to subroutine
          .
          .
          .
SUBR
          .
          .
          .
        MOVR    R5, R7      ; return via addr in R5
```

NOTE: The Series 1600 assemblers recognize JR r as the equivalent of MOVR r, R7.

Subroutine call with argument passed via a general register:

```
        MVI     J, R0       ; fetch argument
        JSR     R4, SUBR    ; transfer to subroutine
          .
          .
SUBR      .
          .
          .
          .
        JR      R4          ; return via R4
```

Subroutine call with argument value following the JSR instruction:

```
        JSR     R5, SUBR    ; transfer to SUBR
        WORD    6           ; arg 1 value
        WORD    2           ; arg 2 value
          .
          .
          .
SUBR    MVI@    R5, R0      ; fetch arg 1 & update return addr
        MVI@    R5, R1      ; fetch arg 2 & update return addr
          .
          .
          .
          .
          .
        JR      R5          ; return
```

Subroutine call with argument address following the JSR instruction; ie, call SUBR (I):

```
        JSR     R5, SUBR    ; transfer to SUBR
        WORD    I           ; address of variable
          .
          .
          .
          .
SUBR    MVI@    R5, R1      ; fetch arg addr, update return
        MVI@    R1, R1      ; fetch value
          .
          .
          .
        JR      R5          ; return
```

Nested subroutines:

```
        JSR     R5, SUB 1   ; transfer to SUB 1
SUB 1   PSHR    R5          ; save return addr on stack
          .
          .
          .
        JSR     R5, SUB 2   ; transfer to next subroutine level
          .
          .
          .
        PULR    PC          ; exit subroutine SUB 1
SUB 2   PSHR    R5          ; save return addr on stack
          .
          .
          .
        JSR     R5, SUB 3   ; transfer to next subroutine level
          .
          .
        PULR    PC          ; exit subroutine SUB 2
SUB 3   .
          .
          .
          .
          .
```

3.4 Program Examples (cont.)

3.4.9 BEXT INSTRUCTION (POLLING)

The BEXT instruction allows up to 16 external conditions to be easily tested by software. These conditions can be switches on a control panel, operator command inputs, data ready signals from peripherals, or any other type of digital status signal. This instruction allows polling schemes to be easily implemented.

Poll inputs and vector to service routines:

```
LOOP BEXT   SRV1, 1      ; test condition 1
     BEXT   SRV2, 2      ; test condition 2
     BEXT   SRV3, 3      ; test condition 3
      .
      .
      .
     B      LOOP         ; branch back to start again
```

Wait loop for I/O device:

```
LOOP BEXT   LOOP, 5      ; test data ready & wait
     MVI    IODATA, R0   ; get data when ready
      .
      .
      .
      .
```

3.5 Instruction Set (Summary listing)

		Mnemonics	Operation	Microcycles				Comments
INTERNAL REFERENCE INSTRUCTIONS	Register to Register	MOVR	MOVe Register	6/7				
		TSTR	TeST Register	6/7				MOVR to itself
		JR	Jump to address in Register	7				MOVR to PC
		ADDR	ADD contents of Registers	6				
		SUBR	SUBtract contents of Registers	6				
		CMPR	CoMPare Registers by subtr.	6				Results not stored
		ANDR	logical AND Registers	6				
		XORR	eXclusive OR Registers	6				
		CLRR	CLeaR Register	6				XORR with itself
	Single Register	INCR	INCrement Register	6				
		DECR	DECrement Register	6				
		COMR	COMplement Register	6				One's Complement
		NEGR	NEGate Register	6				Two's Complement
		ADCR	ADd Carry Bit to Register	6				
		GSWD	Get Status WorD	6				
		NOP	No OPeration	6				
		SIN	Software INterrupt	6				Pulse to $\overline{\text{PCIT}}$ pin
		RSWD	Return Status WorD	6				
	Register Shift	SWAP	SWAP 8-bit bytes	6				Not interruptable. One or two position shift capability. Two position SWAP not supported.
		SLL	Shift Logical Left	6				
		RLC	Rotate Left thru Carry	6				
		SLLC	Shift Logical Left thru Carry	6				
		SLR	Shift Logical Right	6				
		SAR	Shift Arithmetic Right	6				
		RRC	Rotate Right thru Carry	6				
		SARC	Shift Arithmetic Right thru Carry	6				
	Control Instructions	HLT	HaLT	4				
		SDBD	Set Double Byte Data	4				Must precede external reference to double byte data
		EIS	Enable Interrupt System	4				
		DIS	Disable Interrupt System	4				
		TCI	Terminate Current Interrupt	4				Not interruptable
		CLRC	CLeaR Carry to zero	4				
		SETC	SET Carry to one	4				
	Jump Instructions	J	Jump	12				
		JE	Jump, Enable, interrupt	12				
		JD	Jump, Disable interrupt	12				
		JSR	Jump, Save Return	12				Return Address saved in R4, 5 or 6.
		JSRE	Jump, Save Return & Enable	12				
		JSRD	Jump, Save Return & Disable interrupt	12				
EXTERNAL REFERENCE INSTRUCTIONS	Conditional Branch Instructions	B	unconditional Branch	9*				Two words
		NOPP	No OPeration	7*				
		BC (BLGE)	Branch on Carry	7				C=1
		BNC (BLLT)	Branch on No Carry	7				C=0
		BOV	Branch on OVerflow	7				OV=1
		BNOV	Branch on No OVerflow	7				OV=0
		BPL	Branch on PLus	7				S=0
		BMI	Branch on Minus	7				S=1
		BZE (BEQ)	Branch on ZEro or EQual	7				Z=1
		BNZE (BNEQ)	Branch if Not ZEro or Not EQual	7				Z=0
		BLT	Branch if Less Than	7				S∀OV=1
		BGE	Branch if Greater than or Equal	7				S∀OV=0
		BLE	Branch if Less than or Equal	7				ZV(S∀OV)=1
		BGT	Branch if Greater Than	7				ZV(S∀OV)=0
		BUSC	Branch if Sign≠Carry	7				C∀S=1
		BESC	Branch if Sign=Carry	7				C∀S=0
		BEXT	Branch if External condition is True	7				4 LSB of instruction are decoded select 1 of 16 external conditions.
				Add 2 cycles if test condition is true, except *				Displacement in PC+1 PC←PC±Displacement
				Dir.	**Imm.**	**Indir.**	**Stack**	
	I/O	MVO	MoVe Out	11	9	9	9	Not interruptable
		PSHR	PuSH Register to stack	—	—	—	9	PSHR=MVO@R6. Not interruptable
		MVI	MoVe In	10	8	8	11	
		PULR	PULl from stack to Register	—	—	—	11	PULR=MVI@R6.
	Arithmetic & Logic	ADD	ADD	10	8	8	11	
		SUB	SUBtract	10	8	8	11	
		CMP	CoMPare	10	8	8	11	Result not saved
		AND	logical AND	10	8	8	11	
		XOR	eXclusive OR	10	8	8	11	

1 MICROCYCLE=2 CLOCK CYCLES

3.6.1 REGISTER TO REGISTER

Mnemonic	Operand	Cycles	Instruction	Description	Status Change
MOVR	SSS, DDD	6/7*	X XXX XX0 010 SSS DDD	MOVe contents of Register SSS to register DDD.	S, Z
TSTR	SSS	6/7**	0 010 SSS SSS	TeST contents of Register SSS.	S, Z
JR	SSS	7	0 010 SSS 111	Jump to address in Register SSS (move address to Register 7).	S, Z
ADDR	SSS, DDD	6	0 011 SSS DDD	ADD contents of Register SSS to contents of register DDD. Results to DDD.	S, Z, C, OV
SUBR	SSS, DDD	6	0 100 SSS DDD	SUBtract contents of Register SSS from contents of register DDD. Results to DDD.	S, Z, C, OV
CMPR	SSS, DDD	6	0 101 SSS DDD	CoMPare Register SSS with register DDD by subtraction. Results not stored.	S, Z, C, OV
ANDR	SSS, DDD	6	0 110 SSS DDD	logical AND contents of Register SSS with contents of register DDD. Results to DDD.	S, Z
XORR	SSS, DDD	6	0 111 SSS DDD	eXclusive OR contents of Register SSS with contents of register DDD. Results to DDD.	S, Z
CLRR	DDD	6	0 111 DDD DDD	CLeaR Register to zero.	S, Z

*7 Cycles if DDD=6 or 7.
**7 Cycles if SSS=6 or 7.

3.6.2 SINGLE REGISTER

Mnemonic	Operand	Cycles	Instruction	Description	Status Change
INCR	DDD	6	X XXX XX0 000 001 DDD	INCrement contents of Register DDD. Results to DDD.	S, Z
DECR	DDD	6	0 000 010 DDD	DECrement contents of Register DDD. Results to DDD.	S, Z
COMR	DDD	6	0 000 011 DDD	one's COMplement contents of Register DDD. Results to DDD.	S, Z
NEGR	DDD	6	0 000 100 DDD	two's complement contents of Register DDD. Results to DDD.	S, Z, C, OV
ADCR	DDD	6	0 000 101 DDD	ADd Carry bit to contents of Register DDD. Results to DDD.	S, Z, C, OV
GSWD	DD	6	0 000 110 0DD	Get Status WorD in register DD. Bits 0-3, 8-11 set to 0. Bits 4, 12=C;5, 13=OV;6, 14=Z;7, 15=S.	
NOP		6	0 000 110 10X	No Operation.	
SIN		6	0 000 110 11X	Software Interrupt; pulse to $\overline{\text{PCIT}}$ pin.	
RSWD	SSS	6	0 000 111 SSS	Restore Status WorD from register SSS; Bit 4 to C, Bit 5 to OV, Bit 6 to Z, Bit 7 to S.	S, Z, C, OV

3.6.3 REGISTER SHIFT

Executable only with Registers 0, 1, 2, 3.
Shifts are not interruptable.

Mnemonic	Operand	Cycles	Instruction	Description	Status Change
SWAP	RR<,n>	6	X XXX XX0 001 000 NRR	N=0, SWAP bytes of register RR. S equals Bit 7 of results of SWAP.	S, Z
		8		N=1, not supported.	S, Z
SLL	RR<,n>	6	0 001 001 NRR	N=0. Shift Logical Left one bit. Zero to low bit.	S, Z
		8		N=1, Shift Logical Left two bits. Zero to low 2 bits.	S, Z
RLC	RR<,n>	6	0 001 010 NRR	N=0, Rotate Left one bit using C as bit 16.	S, Z, C
		8		N=1, Rotate Left two bits using C as bit 17 and OV as bit 16.	S, Z, C, OV
SLLC	RR<,n>	6	0 001 011 NRR	N=0, Shift Logical Left one bit using C as bit 16. Zero to low bit.	S, Z, C
		8		N=1, Shift Logical Left two bits using C as bit 17, OV as bit 16. Zero to low 2 bits.	S, Z, C, OV
SLR	RR<,n>	6	0 001 100 NRR	N=0, Shift Logical Right one bit. Zero to high bit.	S, Z
		8		N=1, Shift Logical Right two bits. Zero to high two bits.	S, Z
SAR	RR<,n>	6	0 001 101 NRR	N=0, Shift Arithmetic Right one bit. Sign bit copied to high bit.	S, Z
		8		N=1, Shift Arithmetic Right two bits. Sign bit copied to high 2 bits.	S, Z
RRC	RR<,n>	6	0 001 110 NRR	N=0, Rotate right one bit using C as bit 16.	S, Z, C
		8		N=1, Rotate right two bits using C as bit 16 and OV as bit 17.	S, Z, C, OV
SARC	RR<,n>	6	0 001 111 NRR	N=0, Shift Arithmetic Right one bit, thru C. Sign bit copied to high bit.	S, Z, C
		8		N=1, Shift Arithmetic right two bits, thru OV and C. Sign bit copied to high 2 bits.	S, Z, C, OV

NOTE: n=1 or 2 (places).

3.6.4 CONTROL

Mnemonic	Operand	Cycles	Instruction	Description	Status Change
HLT		4	X XXX XX0 000 000 000	HaLT after next interruptable instruction is executed. Resume on start.	
SDBD		4	0 000 000 001	Set Double Byte Data for the next instruction which must be an external reference instruction.	
EIS		4	0 000 000 010	Enable Interrupt System. Not Interruptable.	
DIS		4	0 000 000 011	Disable Interrupt System. Not Interruptable.	
TCI		4	0 000 000 101	Terminate Current Interrupt. Not Interruptable.	
CLRC		4	0 000 000 110	CLeaR Carry to zero. Not Interruptable.	C
SETC		4	0 000 000 111	SET Carry to one. Not Interruptable.	C

3.6.5 JUMP

Mnemonic	Operand	Cycles	Instruction	Description	Status Change
J	DA	12	X XXX XX0 000 000 100 X XXX XX1 1AA AAA A00 X XXX XXA AAA AAA AAA	Jump to address. Program counter is set to 16 bits of A's.	
JE	DA	12	0 000 000 100 1 1AA AAA A01 A AAA AAA AAA	Jump to address. Enable interrupt system. Program counter is set to 16 bits of A's.	
JD	DA	12	0 000 000 100 1 1AA AAA A10 A AAA AAA AAA	Jump to address. Disable interrupt system. Program counter is set to 16 bits of A's.	
JSR	BB, DA	12	0 000 000 100 B BAA AAA A00 A AAA AAA AAA	Jump and Save Return address (PC+3) in register designated by 1BB. Program counter is set to 16 bits of A's. BB≠11.	
JSRE	BB, DA	12	0 000 000 100 B BAA AAA A01 A AAA AAA AAA	Jump and Save Return and Enable interrupt system. Return (PC+3) is saved in register 1BB. Program counter is set to 16 bits of A's. BB≠11.	
JSRD	BB, DA	12	0 000 000 100 B BAA AAA A10 A AAA AAA AAA	Jump and Save Return and Disable interrupt system. Return (PC+3) is saved in register 1BB. Program counter is set to 16 bits of A's. BB≠11.	

NOTE: Bits 2-7 of the second word form bits 10-15 of the Destination Address.
Bits 0-9 of the third word form bits 0-9 of the Destination Address.

3.6.6 BRANCHES

The Branch instructions are Program Counter Relative, i.e., the Effective Address=PC ±Displacement.
P-P is the Displacement and S is 0 for +, 1 for −.
For a forward branch an addition is performed.
For a backward branch a one's complement subtraction is performed.
Computation performed on PC+2.

Mnemonic	Operand	Cycles	Instruction	Description	Status Change
B	DA	9	X XXX XX1 000 S00 000 P PPP PPP PPP PPP PPP	Branch unconditional. Program Counter Relative (+1025 to −1024)	
NOPP		7	1 000 S01 000 P PPP PPP PPP PPP PPP	No OPeration, two words	
BC (BLGE)	DA	7/9	1 000 S00 001 P PPP PPP PPP PPP PPP	Branch on Carry (Branch if Logical Greater Than or Equal). C=1.	
BNC (BLLT)	DA	7/9	1 000 S01 001 P PPP PPP PPP PPP PPP	Branch on No Carry (Branch if Logical Less Than). C=0	
BOV	DA	7/9	1 000 S00 010 P PPP PPP PPP PPP PPP	Branch on OVerflow. OV=1	
BNOV	DA	7/9	1 000 S01 010 P PPP PPP PPP PPP PPP	Branch on No OVerflow. OV=0	
BPL	DA	7/9	1 000 S00 011 P PPP PPP PPP PPP PPP	Branch on PLus. S=0	
BMI	DA	7/9	1 000 S01 011 P PPP PPP PPP PPP PPP	Branch on MInus. S=1	
BZE (BEQ)	DA	7/9	1 000 S00 100 P PPP PPP PPP PPP PPP	Branch on ZEro (Branch if EQual). Z=1	
BNZE (BNEQ)	DA	7/9	1 000 S01 100 P PPP PPP PPP PPP PPP	Branch on No ZEro (Branch if Not EQual). Z=0	
BLT	DA	7/9	1 000 S00 101 P PPP PPP PPP PPP PPP	Branch if Less Than. S∀OV=1	
BGE	DA	7/9	1 000 S01 101 P PPP PPP PPP PPP PPP	Branch if Greater than or Equal. S∀OV=0	
BLE	DA	7/9	1 000 S00 110 P PPP PPP PPP PPP PPP	Branch if Less than or Equal. ZV (S∀OV)=1	
BGT	DA	7/9	1 000 S01 110 P PPP PPP PPP PPP PPP	Branch if Greater Than. ZV (S∀OV)=0	
BUSC	DA	7/9	1 000 S00 111 P PPP PPP PPP PPP PPP	Branch if Unequal Sign and Carry C∀S=1	
BESC	DA	7/9	1 000 S01 111 P PPP PPP PPP PPP PPP	Branch if Equal Sign and Carry C∀S=0	
BEXT	DA, EEEE	7/9	1 000 S1E EEE P PPP PPP PPP PPP PPP	Branch if EXternal condition is True. Field EEEE is externally decoded to select 1 of 16 conditions. Response is tested for true condition.	

NOTE: 7/9=7 Cycles if test condition is not true, 9 cycles if true.

3.6.7 DIRECT ADDRESSED DATA—MEMORY

Mnemonic	Operand	Cycles	Instruction	Description	Status Change
MVO	SSS, DA	11	X XXX XX1 001 000 SSS A AAA AAA AAA AAA AAA	MoVe Out data from register SSS to address A-A. Not interruptable.	
MVI	SA, DDD	10	1 010 000 DDD A AAA AAA AAA AAA AAA	MoVe In data from address A-A to register DDD.	
ADD	SA, DDD	10	1 011 000 DDD A AAA AAA AAA AAA AAA	ADD data from address A-A to register DDD. Results to DDD.	S, Z, C, OV
SUB	SA, DDD	10	1 100 000 DDD A AAA AAA AAA AAA AAA	SUBtract data from address A-A from register DDD. Results to DDD.	S, Z, C, OV
CMP	SA, SSS	10	1 101 000 SSS A AAA AAA AAA AAA AAA	CoMPare data from address A-A with register SSS by subtraction. Results not stored.	S, Z, C, OV
AND	SA, DDD	10	1 110 000 DDD A AAA AAA AAA AAA AAA	logical AND data from address A-A with register DDD. Results to DDD.	S, Z
XOR	SA, DDD	10	1 111 000 DDD A AAA AAA AAA AAA AAA	eXclusive OR data from address A-A with register DDD. Results to DDD.	S, Z

3.6.8 IMMEDIATE DATA—REGISTER

Mnemonic	Operand	Cycles	Instruction	Description	Status Change
MVOI	SSS, I-I	9	X XXX XX1 001 111 SSS I III III III III III	MoVe Out Immediate data from register SSS to PC+1 (field). Not interruptable.	
MVII	I-I, DDD	8	1 010 111 DDD I III III III III III	MoVe In Immediate data to register DDD from PC+1 (field).	
ADDI	I-I, DDD	8	1 011 111 DDD I III III III III III	Add Immediate data to contents of register DDD. Results to DDD.	S, Z, C, OV
SUBI	I-I, DDD	8	1 100 111 DDD I III III III III III	SUBtract Immediate data from contents of register DDD. Results to DDD.	S, Z, C, OV
CMPI	I-I, SSS	8	1 101 111 SSS I III III III III III	CoMPare Immediate data from contents of register SSS by subtraction. Results not stored.	S, Z, C, OV
ANDI	I-I, DDD	8	1 110 111 DDD I III III III III III	logical AND Immediate data with contents of register DDD. Results to DDD.	S, Z
XORI	I-I, DDD	8	1 111 111 DDD I III III III III III	eXclusive OR Immediate data with contents of register DDD. Results to DDD.	S, Z

3.6.9 INDIRECT ADDRESSED DATA—REGISTER

MMM Source data is located at the address contained in Register R1-R6.
MMM=4, 5: post-increment R4 or R5.
MMM=6: MVO instruction—post-increment R6. PUSH data from Register SSS to the Stack.
Other instructions—pre-decrement R. PULL data from the Stack to be used as the first operand.

Mnemonic	Operand	Cycles	Instruction	Description	Status Change
MVO@	SSS, MMM	9	X XXX XX1 001 MMM SSS	MoVe Out data from register SSS to the address in register MMM. Note: SSS=MMM=4, 5, 6 or 7 not supported. Not interruptable.	
PSHR	SSS	9	1 001 110 SSS	PuSH data from Register SSS to the stack. Not interruptable.	
MVI@	MMM, DDD	8/11	1 010 MMM DDD	MoVe In data from the address in register MMM to register DDD.	
PULR	DDD	11	1 010 110 DDD	PULl data from the stack to Register DDD.	
ADD@	MMM, DDD	8/11	1 011 MMM DDD	ADD data located at the address in register MMM to contents of register DDD. Results to DDD.	S, Z, C, OV
SUB@	MMM, DDD	8/11	1 100 MMM DDD	SUBtract data located at the address in register MMM from contents of register DDD. Results to DDD.	S, Z, C, OV
CMP@	MMM, DDD	8/11	1 101 MMM SSS	CoMPare data located at the address in register MMM with contents of register SSS by subtraction. Results not stored.	S, Z, C, OV
AND@	MMM,DDD	8/11	1 110 MMM DDD	logical AND contents of register DDD with data located at the address in register MMM. Results to DDD.	S, Z
XOR@	MMM, DDD	8/11	1 111 MMM DDD	eXclusive OR contents of register DDD with data located at the address in register MMM. Results to DDD.	S, Z

NOTE: 8/11=11 Cycles if MMM=6, 8 Cycles otherwise.

3.6.10 IMMEDIATE DOUBLE BYTE DATA—REGISTER

NOTE: The SDBD command is provided by the assembler when the immediate data is greater than the specified memory width and requires two bytes.

Mnemonic	Operand	Cycles	Instruction	Description	Status Change
SDBD			X XXX XX0 000 000 001	MoVe In Immediate double byte data to register DDD.	
MVII	I-I, DDD	14	X XXX XX1 010 111 DDD		
			X XXX XXX XLL LLL LLL		
			X XXX XXX XUU UUU UUU		
SDBD			0 000 000 001	ADD Immediate double byte data to contents of register DDD. Results to DDD.	S, Z, C, OV
ADDI	I-I, DDD	14	1 011 111 DDD		
			LL LLL LLL		
			UU UUU UUU		
SDBD			0 000 000 001	SUBtract Immediate double byte data from contents of register DDD. Results to DDD.	S, Z, C, OV
SUBI	I-I, DDD	14	1 100 111 DDD		
			LL LLL LLL		
			UU UUU UUU		
SDBD			0 000 000 001	CoMPare Immediate double byte data with contents of register SSS by subtraction. Results not stored.	S, Z C, OV
CMPI	I-I, SSS	14	1 101 111 SSS		
			LL LLL LLL		
			UU UUU UUU		
SDBD			0 000 000 001	logical AND Immediate double byte data with contents of register DDD. Results to register DDD.	S, Z
ANDI	I-I, DDD	14	1 110 111 DDD		
			LL LLL LLL		
			UU UUU UUU		
SDBD			0 000 000 001	eXclusive OR Immediate double byte data with contents of register DDD. Results to register DDD.	S, Z
XORI	I-I, DDD	14	1 111 111 DDD		
			LL LLL LLL		
			UU UUU UUU		

NOTE: I-I=UUUUUUUULLLLLLLL; L-L indicates low byte of literal; U-U indicates upper byte.

3.6.11 INDIRECT ADDRESSED DOUBLE BYTE DATA—REGISTER

NOTE: The SDBD instruction is normally supplied by the assembler as required to properly generate machine code.

Mnemonic	Operand	Cycles	Instruction	Description	Status Change
SDBD			X XXX XX0 000 000 001	MoVe In double byte data from the address in register MMM to register DDD.	
MVI@	MMM, DDD	14	X XXX XX1 010 MMM DDD		
SDBD			0 000 000 001	ADD double byte data located at the address in register MMM to contents of register DDD. Results to DDD.	S, Z, C, OV
ADD@	MMM, DDD	14	1 011 MMM DDD		
SDBD			0 000 000 001	SUBtract double byte data located at the address in register MMM from contents of register DDD. Results to DDD.	S, Z, C OV
SUB@	MMM, DDD	14	1 100 MMM DDD		
SDBD			0 000 000 001	CoMPare double byte data located at the address in register MMM with contents of register SSS by subtraction. Results not stored.	S, Z, C, OV
CMP@	MMM, DDD	14	1 101 MMM SSS		
SDBD			0 000 000 001	logical AND double byte data located at the address in register MMM with contents of register DDD. Results to DDD.	S, Z
AND@	MMM, DDD	14	1 110 MMM DDD		
SDBD			0 000 000 001	eXclusive OR double byte data located at the address in register MMM with contents of register DDD. Results to DDD.	S, Z
XOR@	MMM, DDD	14	1 111 MMM DDD		

3.6.12 SYMBOLIC NOTATION—The following symbolic notation is used in all CP1600 instruction documentation.

Address Modes:
- R - register
- 'blank' - direct address
- I - immediate data
- @ - indirect address

Operands:
- SSS - Source Register
- DDD - Destination Register
- MMM - Register Address Mode
- RR - Register (0-3)
- N - Number of Shifts (0=1, 1=2)
- S - Sign of Address Displacement
- EEEE - External Condition Code (0-15)
- DA - Destination Address
- SA - Source Address
- I-I - Immediate data word
- P-P - Address displacement for Branch

Functions:
- + - addition
- − - subtraction
- V - inclusive OR
- ∀ - exclusive OR
- Λ - AND
- () - contents of
- ← - is replaced by
- <> - optional operand

Status:
- S - Sign bit
- Z - Zero bit
- C - Carry bit
- OV - OVerflow bit

MMM - Register Address Mode
- 000 - direct address in location following instruction.
- 001 - indirect address for Register 1
- 010 - indirect address for Register 2
- 011 - indirect address for Register 3
- 100 - indirect address for Register 4, post increment
- 101 - indirect address for Register 5, post increment
- 110 - indirect address for Register 6, post increment for MVO only; indirect address for Register 6, pre decrement for all instructions except MVO.
- 111 - indirect address for Register 7, post increment. (Immediate data in location following instruction.)

4 SYSTEM IMPLEMENTATION

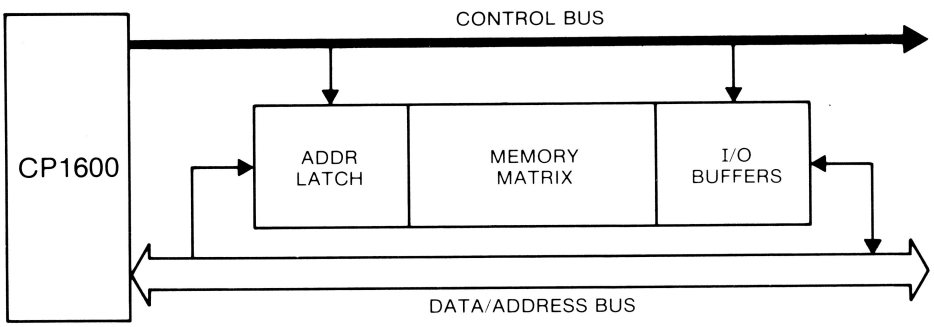
The CP1600 is a single chip, 16-bit MOS-LSI microprocessor array with many advance features enabling its use as the heart of a high performance microcomputer-based system. As such, it must be configured to meet both the processing and economic requirements of its end product environment. The following sections discuss a number of ways to configure the CP1600 microprocessor into a working system.

4.1 Bus Structure

The bus structure of the CP1600 microprocessor array is based on a single 16-bit bidirectional bus used to output both data and addresses and input instructions, data, and interrupt addresses. However, in a microcomputer based product there are a number of different approaches to implementing the total system bus structure around the single bidirectional bus of the CP1600.

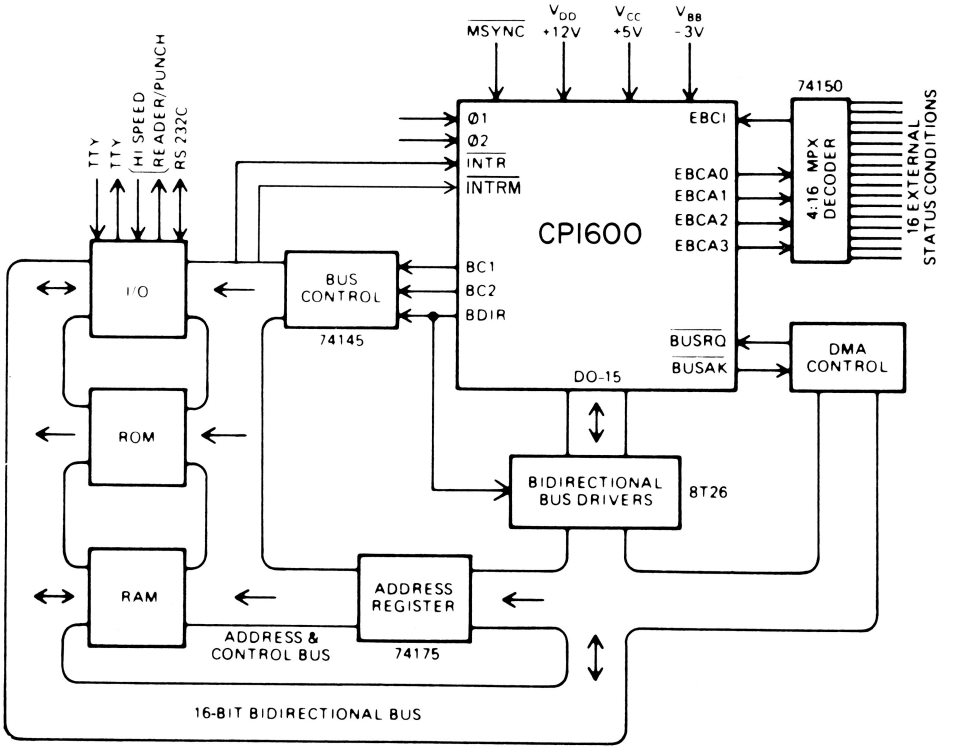
In small systems with only a few memory chips, limited peripheral interfacing, and minimal expansion capability, the CP1600 bus can be used directly without buffering. To accommodate the unbuffered, single bus approach, many of the General Instrument ROM and RAM products contain internal address and chip select registers to trap and hold memory addresses from the CP1600 bus. This reduces parts count significantly in these small systems. In addition, a convenient and cost-saving technique in these small systems is to utilize the upper bits of the 16-bit address as direct chip select signals. This creates a non-contiguous memory allocation but eliminates the need for full binary decoding of the upper part of the address field. Because the CP1600 provides a full 16-bit address, there is plenty of address space available to accommodate this technique in small systems.

Fig. 18 SIMPLE SYSTEM IMPLEMENTATION



In systems with larger memories, more sophisticated peripheral interfacing, and extensive modularity and expandability, the CP1600 is best utilized with bipolar buffering of the bidirectional bus. Many standard multiple-sourced TTL parts are presently available for this purpose and provide the system designer with a number of important configuration options.

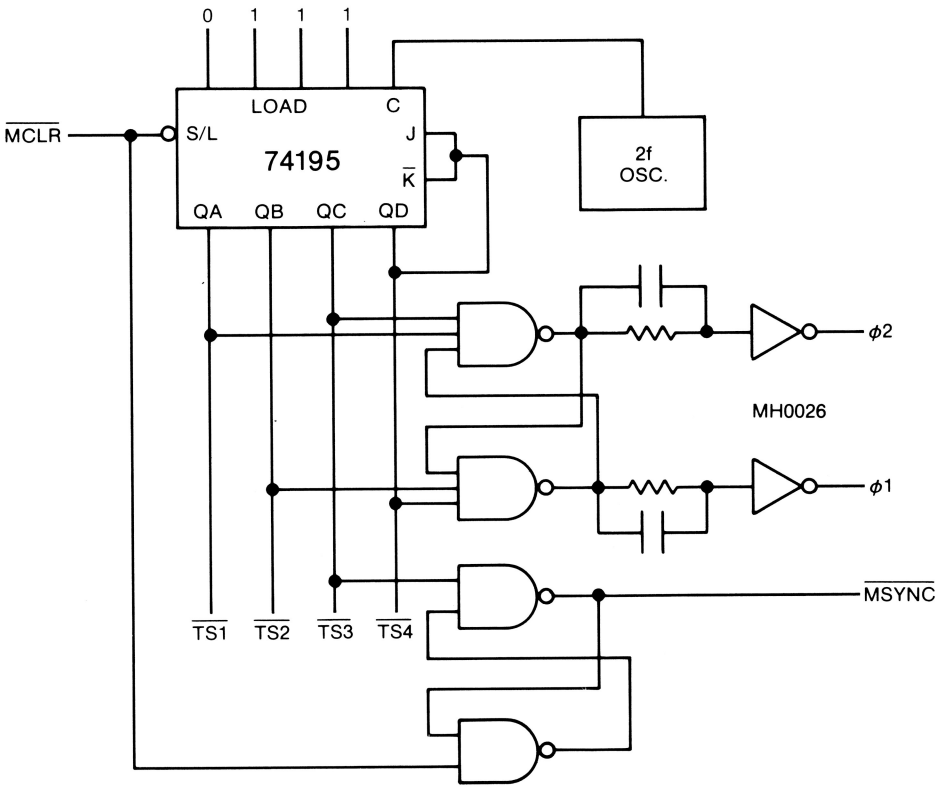
Fig. 19 ADVANCED SYSTEM IMPLEMENTATION



4.2 Clocks

The CP1600 requires a two phase non-overlapping high level clock system for proper operation. Internally, the $\phi 1/\phi 2$ clocks are used to generate the four internal time slots TS1, TS2, TS3, TS4, which make up each microcycle. In addition, the power-on/master clear sequence requires that $\overline{\text{MSYNC}}$ go high at the rising edge of a $\phi 1$ clock phase (TS3). The following example is a typical clock generation circuit:

Fig. 20 TYPICAL CLOCK GENERATION CIRCUIT



4.3 External Sense Logic

The Branch on EXTernal condition (BEXT) instruction of the CP1600 allows up to 16 external digital signals to be sampled by the program with a program branch occurring if the test is true. The four lower bits of the Instruction Register (IR) are presented at pins EBCA0-3 and the CPU samples EBCI to determine if the branch should occur. Two simple examples of the implementation of this function are shown below.

Fig. 21 16 MUTUALLY EXCLUSIVE BRANCH TESTS

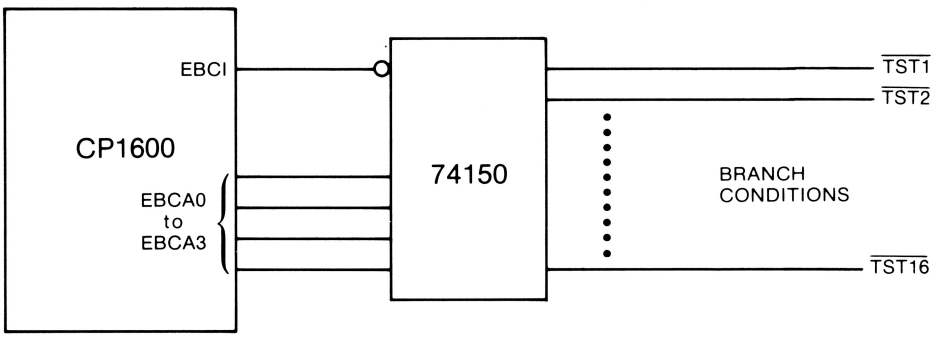
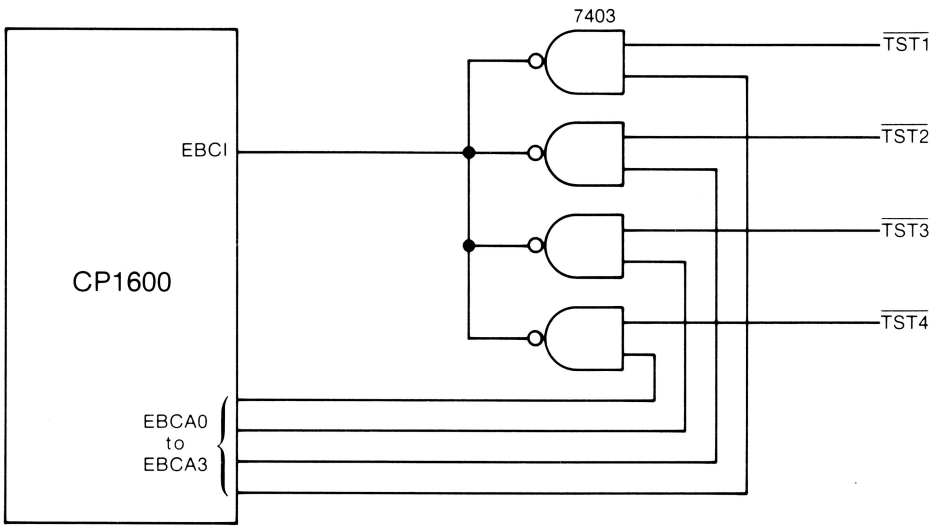


Fig. 22 4 TEST CONDITIONS WITH MULTIPLE "AND" CAPABILITY



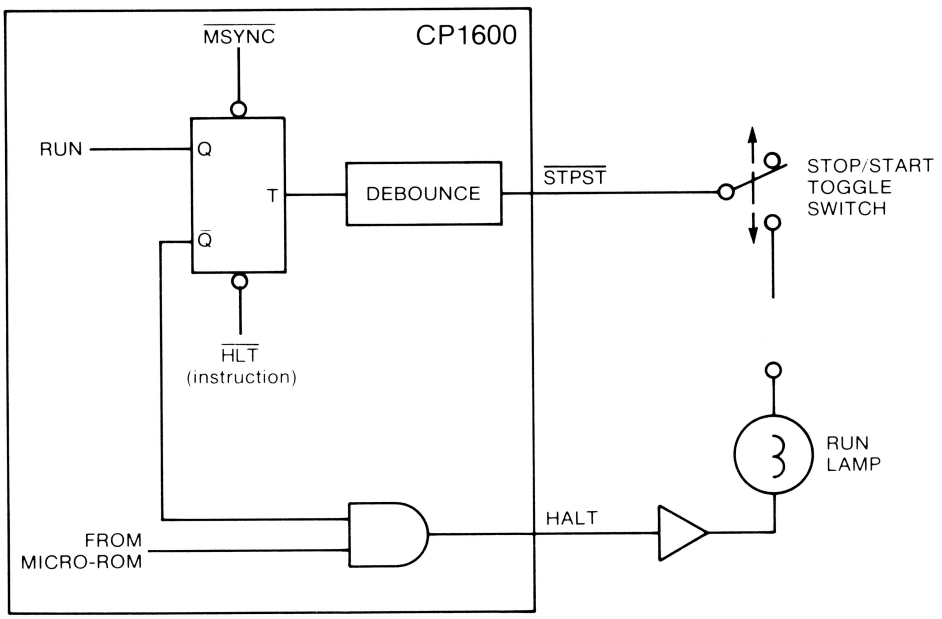
4.4 Start/Stop and Halt

Most systems incorporating a microprocessor require some sort of start/stop mechanism and a simple indication of the present operational mode; i.e., running or stopped. The CP1600 has an internal START/STOP FLIP-FLOP to provide these functions totally on-chip.

The $\overline{\text{STPST}}$ pin is a negative edge trigger to the toggle input of this flip-flop so that alternate pulses cause transitions between the RUN and STOP modes of the CPU. This input also has an internal debounce circuit so that a momentary switch can be directly connected to the CPU to perform the START/STOP function.

The HALT output from the CPU is simply the indication of the RUN/STOP condition of the processor. Because the CPU stops only after the completion of an instruction, the HALT signal is generated from a logical combination of the START/STOP FLIP-FLOP and the microcontrol sequencing logic. It represents the true state of the operational mode of the CP1600.

Fig. 23 CP1600 OPERATIONAL MODE CONTROL/INDICATION



4.5 Interrupt System

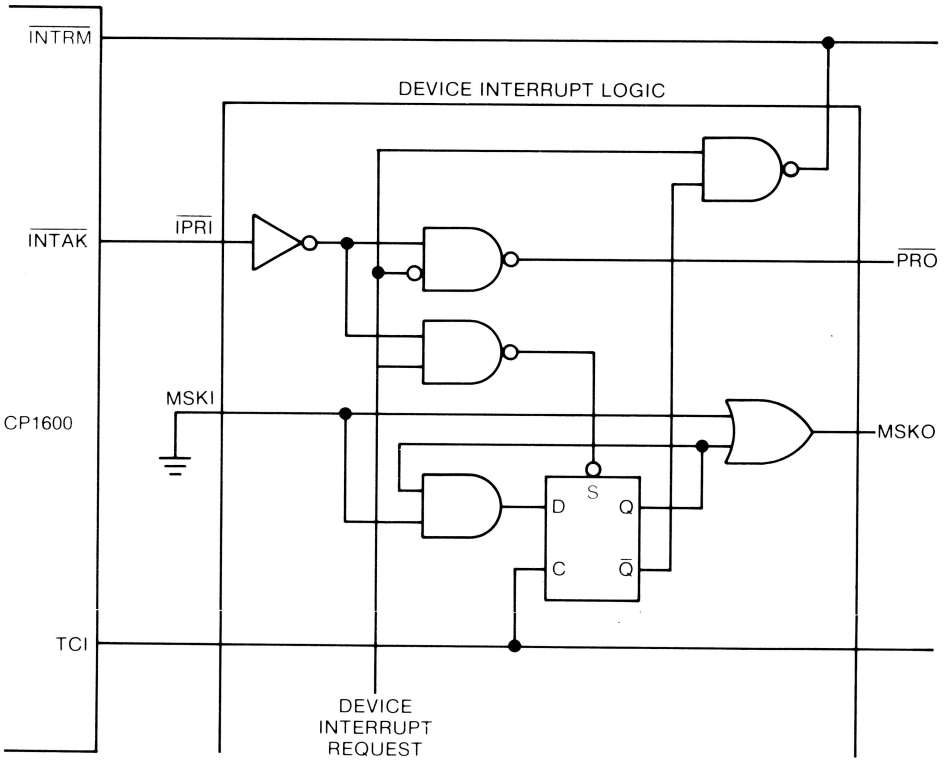
The interrupt system of the CP1600 microprocessor is based on a generalized multi-line/multi-level configuration which can be used to implement interrupt structures ranging in complexity from simple "polled" priority networks to sophisticated fully reentrant nested priority structures. In addition, interrupts can be identified in groups or individually as the system performance demands.

The CP1600 provides five signals to control the interrupt system. These signals include two request lines, $\overline{\text{INTR}}$ and $\overline{\text{INTRM}}$, two bus control signals, INTAK and IAB, and an interrupt termination signal, TCI. In the simplest case, each device can request interrupt service on the $\overline{\text{INTRM}}$ line with a fixed interrupt address vector being generated during the IAB signal. The software service program can then disable the $\overline{\text{INTRM}}$ line and "poll" via BEXT instructions to determine the highest priority device requiring service. At the completion of the appropriate service, the $\overline{\text{INTRM}}$ line can be reenabled by the EIS command.

In a more sophisticated interrupt system, the INTAK acknowledge signal can be used to find the highest priority interrupting device via a daisy-chain network. Acknowledgement of the highest priority device masks out all lower priority devices allowing only higher priority devices to generate interrupts and break into device service already in progress. These nested interrupts can be terminated in reverse order by the TCI pulse with the use of a simple daisy-chain network between the ACK FF in each device.

All of the interrupt system signals are discussed fully in Section 2.3.2 and a simple example of a nested interrupt structure is shown below.

Fig. 24 NESTED INTERRUPT STRUCTURE



4.6 Basic Input/ Output Ports

The most fundamental input/output structure of any digital processor is the simple program-controlled I/O PORT. These interfaces are composed of output holding registers and input transfer gates with all peripheral device control performed by software programming.

Fig. 25a I/O PORT—EXAMPLE 1

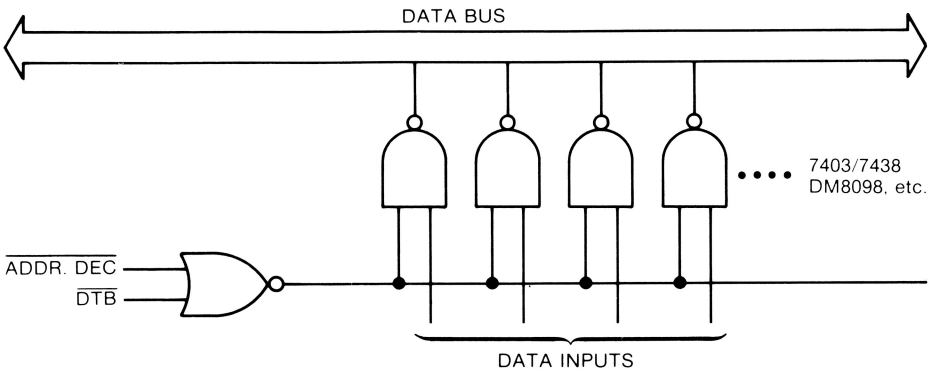


Fig. 25b I/O PORT—EXAMPLE 2

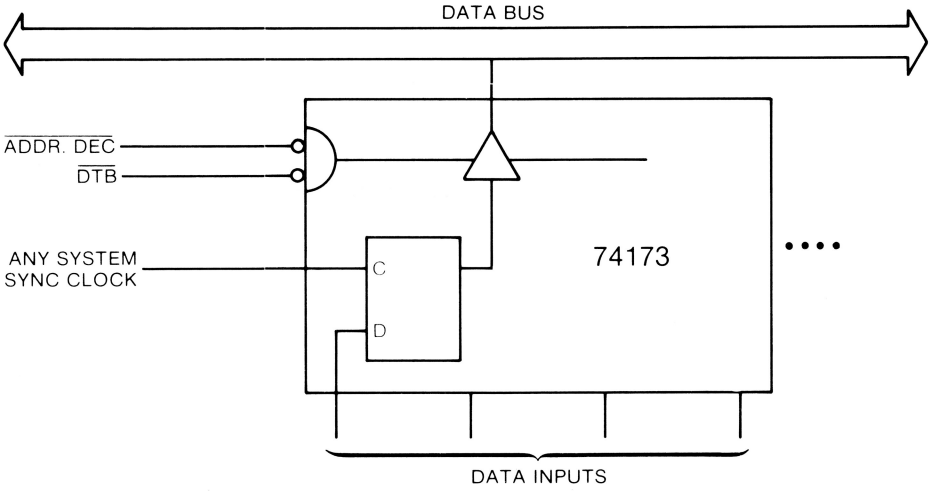
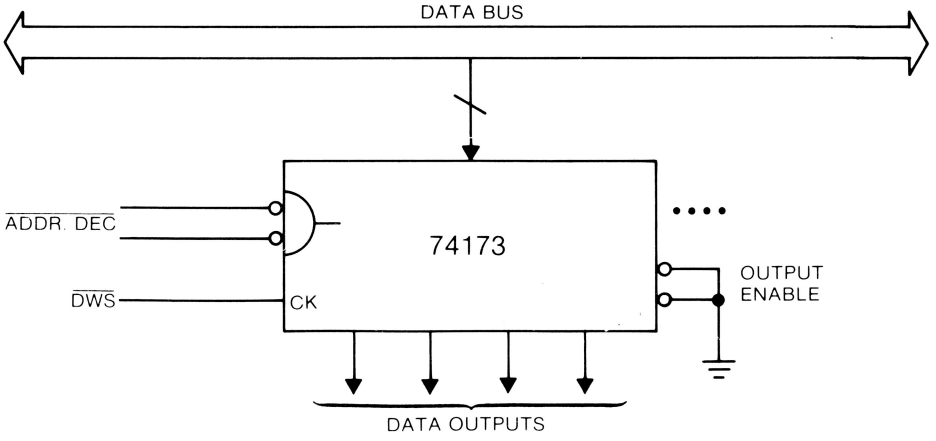


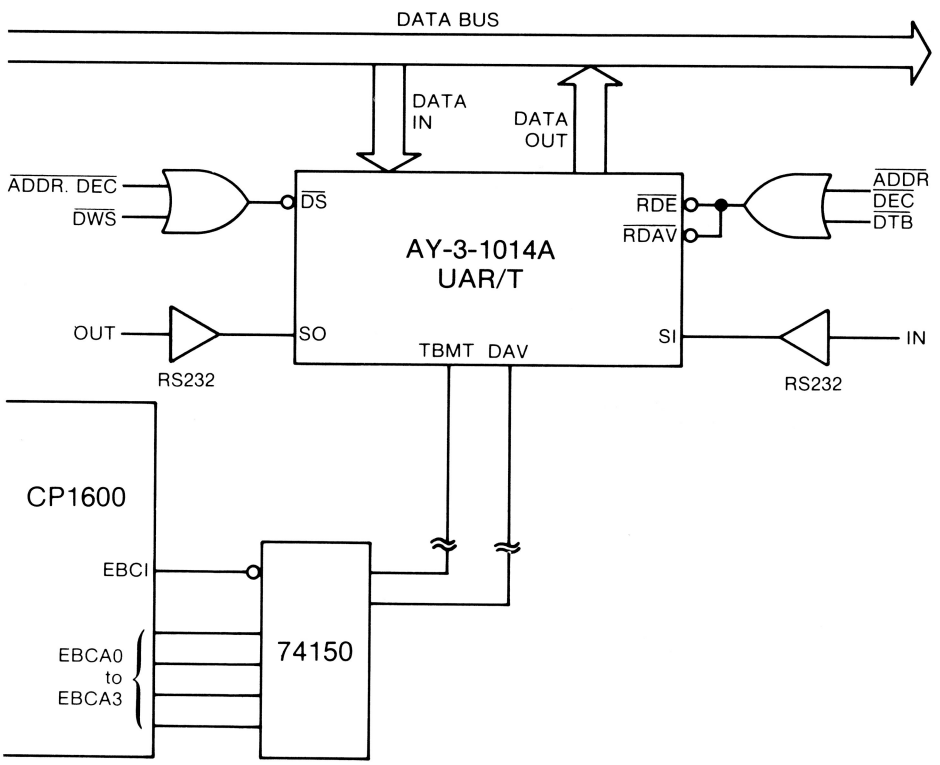
Fig. 25c I/O PORT—EXAMPLE 3



4.7 Simple Communications Interface

The following example shows a low cost, byte buffered communications interface utilizing a Universal Asynchronous Receiver/Transmitter (UAR/T). This interface allows data rates of 60k Baud to be easily handled with very few components. Because the UAR/T is internally double buffered in both the receiver and transmitter sections, the CP1600 program has a full 170 μsec to process characters.

Fig. 26 UAR/T INTERFACE



5 ELECTRICAL SPECIFICATIONS

5.1 Maximum Ratings

V_{DD}, V_{CC}, GND and all other input/output voltages with respect to V_{BB} ..... −0.3V to +18.0V
Storage Temperature ..... −55° C to +150° C
Operating Temperature ..... 0° C to +70° C

Exceeding these ratings could cause permanent damage to these devices. Functional operation at these conditions is not implied—operating conditions are specified in the following paragraphs.

5.2 Standard Conditions

CP1600:
V_{DD}=+12V±5%
V_{CC}=+5V±5%
V_{BB}=−3V±10%
T_A=0° C to +70° C

CP1610:
V_{DD}=+11V±5%
V_{CC}=+5V±5%
V_{BB}=−3V±10%
T_A=0° C to +70° C

5.3 DC Characteristics

Characteristic	Sym	Min	Typ*	Max	Units	Conditions
Clock Inputs						
High	V_{IHC}	10.4	—	V_{DD}	V	
Low	V_{ILC}	0	—	0.6	V	
Input Current:	I_C	—	—	15	mA	$V_{IHC}=(V_{DD}-1)$ V
Logic Inputs						
Low	V_{IL}	0	—	0.65	V	
High (All Lines except BDRDY)	V_{IH}	2.4	—	V_{CC}	V	
High (Bus Data Ready Line See Note)	V_{IHB}	3.0	—	V_{CC}	V	
Logic Outputs						
High	V_{OH}	2.4	—	V_{CC}	V	I_{OH}=100μA
Low (Data Bus Lines D0-D15)	V_{OL}	—	—	0.5	V	I_{OL}=1.6mA
Low (Bus Control Lines BC1, BC2, BDIR)	V_{OL}	—	—	0.45	V	I_{OL}=2.0mA
Low (All Others)	V_{OL}	—	—	0.45	V	I_{OL}=1.6mA
Power Supply Current						
V_{DD} Supply	I_{DD}	—	70	110	mA	
V_{CC} Supply	I_{CC}	—	12	25	mA	
V_{BB} Supply	I_{BB}	—	0.2	2	mA	

*Typical values are at +25° C and nominal voltages.

5.4 AC Characteristics

CP1600: 1 TTL load, 25 pF.

Characteristic	Sym	Min	Typ*	Max	Units	Conditions
Clock Pulse Inputs, $\phi1$ or $\phi2$						
Pulse Width	$t_{\phi1}$, $t_{\phi2}$	120	—	—	ns	Fig. 27
Skew ($\phi1$, $\phi2$ delay)	t_{12}, t_{21}	0	—	—	ns	
Clock Period	t_{CY}	0.3	—	2.0	μs	
Rise & Fall Times	t_r, t_f	—	—	15	ns	
Master SYNC:						
Delay from ϕ	t_{MS}	—	—	30	ns	Fig. 28
D0-D15 Bus Signals						
Output delay from $\phi1$: (float to output)	t_{BO}	—	—	120	ns	Fig. 27
Output delay from $\phi2$: (output to float)	t_{BF}	—	50	—	ns	
Input setup time before $\phi1$	t_{B1}	0	—	—	ns	
Input hold time after $\phi1$	t_{B2}	10	—	—	ns	
Bus Control Signals BC1, BC2, BDIR						
Output delay from $\phi1$	t_{OC}	—	—	120	ns	
Skew	—	—	—	30	ns	
BUSAK Output delay from $\phi1$	t_{BU}	—	150	—	ns	Fig. 29
TCI Output delay from $\phi1$	t_{TO}	—	200	—	ns	Fig. 30
TCI Pulse Width	t_{TW}	—	300	—	ns	
EBCA output delay from BEXT input	t_{DE}	—	—	150	ns	Fig. 31
EBCA wait time for EBCI input	t_{AI}	—	—	400	ns	

*Typical values are at +25°C and nominal voltages.

NOTE: The Bus Data ReaDY (BDRDY) line is sampled during time period TSI after a BAR or ADAR bus control signal. BDRDY must go low requesting a wait state 50 ns before the end of TS1 and remain low for 50 ns minimum. BDRDY may go high asynchronously. In response to DBRDY, the CPU will extend bus cycles by adding additional microcycles up to a maximum of 40 μsec duration.

5.4 AC Characteristics (cont.)

CP1610: 1 TTL load, 100 pF.

Characteristic	Sym	Min	Typ*	Max	Units	Conditions
Clock Pulse Inputs, $\phi1$ or $\phi2$						
Pulse Width	$t_{\phi1}$, $t_{\phi2}$	200	—	—	ns	Fig. 27
Skew ($\phi1$, $\phi2$ delay)	t_{12}, t_{21}	0	—	—	ns	
Clock Period	t_{CY}	0.5	—	2.0	μs	
Rise & Fall Times	t_r, t_f	—	—	15	ns	
Master SYNC:						
Delay from ϕ	t_{MS}	—	—	30	ns	Fig. 28
D0-D15 Bus Signals						
Output delay from $\phi1$ (float to output)	t_{BO}	—	—	100	ns	Fig. 27
Output delay from $\phi2$ (output to float)	t_{BF}	—	50	—	ns	
Input setup time before $\phi1$	t_{B1}	0	—	—	ns	
Input hold time after $\phi1$	t_{B2}	10	—	—	ns	
Bus Control Signals BC1, BC2, BDIR						
Output delay from $\phi1$	t_{OC}	—	—	100	ns	
Skew	—	—	—	30	ns	
BUSAK Output delay from $\phi1$	t_{BU}	—	150	—	ns	Fig. 29
TCI Output delay from $\phi1$	t_{TO}	—	200	—	ns	Fig. 30
TCI Pulse Width	t_{TW}	—	300	—	ns	
EBCA output delay from BEXT input	t_{DE}	—	—	150	ns	Fig. 31
EBCA wait time for EBCI input	t_{AI}	—	—	400	ns	

*Typical values are at +25°C and nominal voltages.

NOTE: The Bus Data ReaDY (BDRDY) line is sampled during time period TSI after a BAR or ADAR bus control signal. BDRDY must go low requesting a wait state 50 ns before the end of TS1 and remain low for 50 ns minimum. BDRDY may go high asynchronously. In response to BDRDY, the CPU will extend bus cycles by adding additional microcycles up to a maximum of 40 μsec duration.

5.5 Capacitance

Characteristic	Sym	Min	Typ	Max	Units	Conditions
$\phi1$, $\phi2$ Clock Input capacitance	$C_{\phi1}$, $C_{\phi2}$	—	20	30	pF	T_A=+25°C, nominal power supplies, $t_{\phi1}$=$t_{\phi2}$=120ns.
D0-D15	—	—	8	15	pF	
All Other	—	—	5	10	pF	

5.6 Timing Diagram References

Fig. 27 CLOCK & BUS TIMING

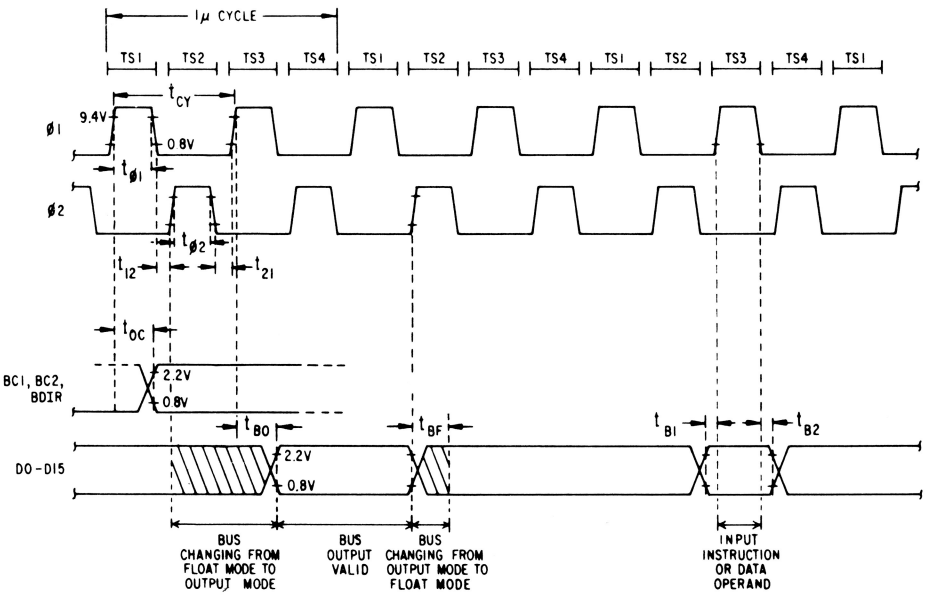


Fig. 28 MASTER SYNC TIMING

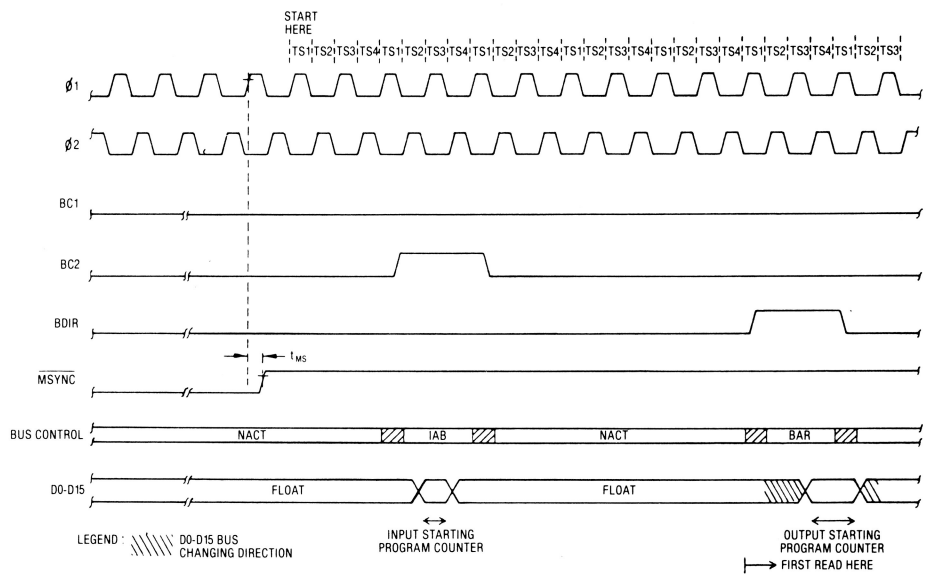


Fig. 29 BUS ACKNOWLEDGE TIMING

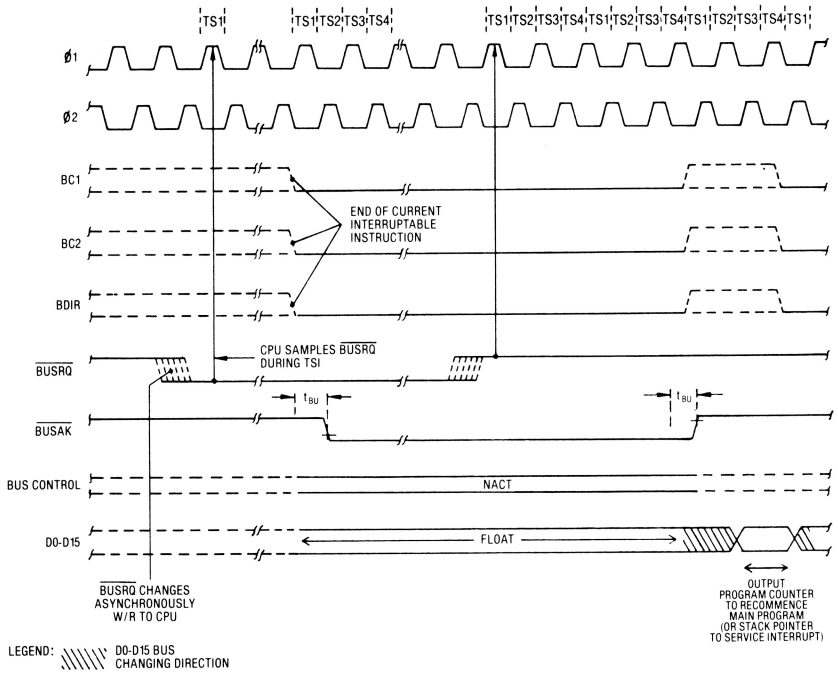


Fig. 30 INTERRUPT TIMING

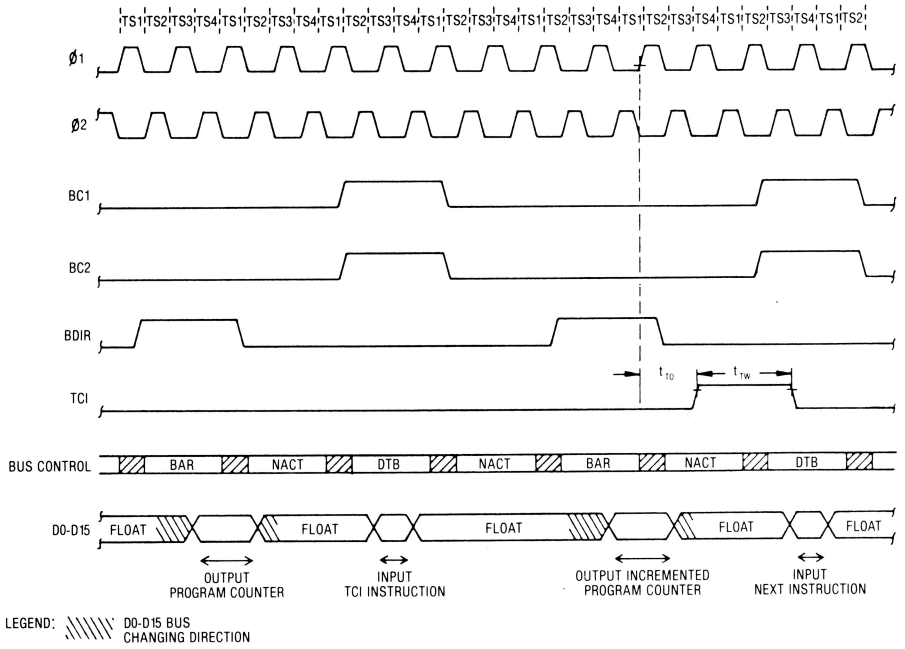
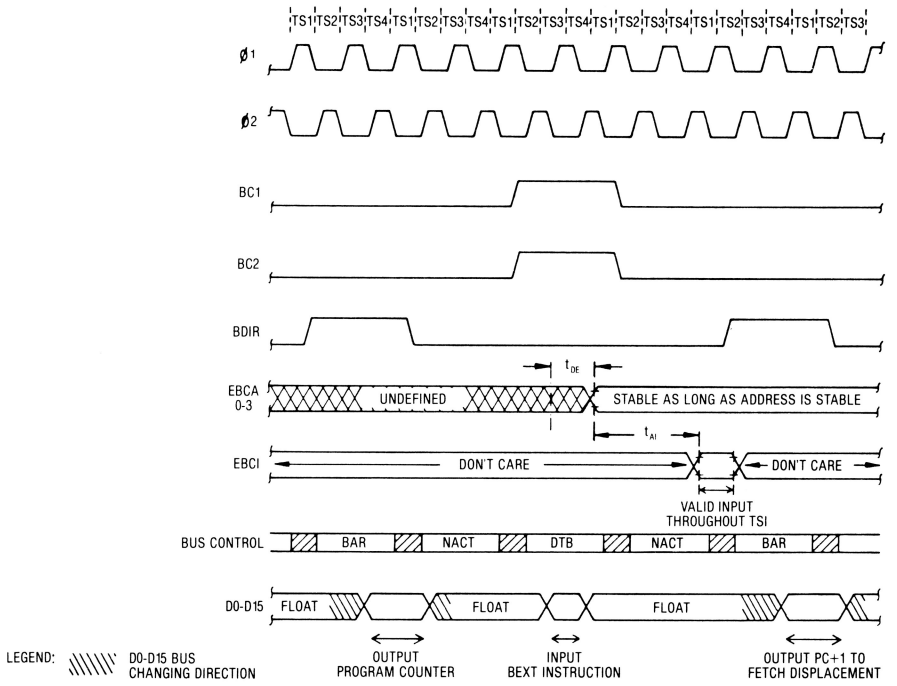
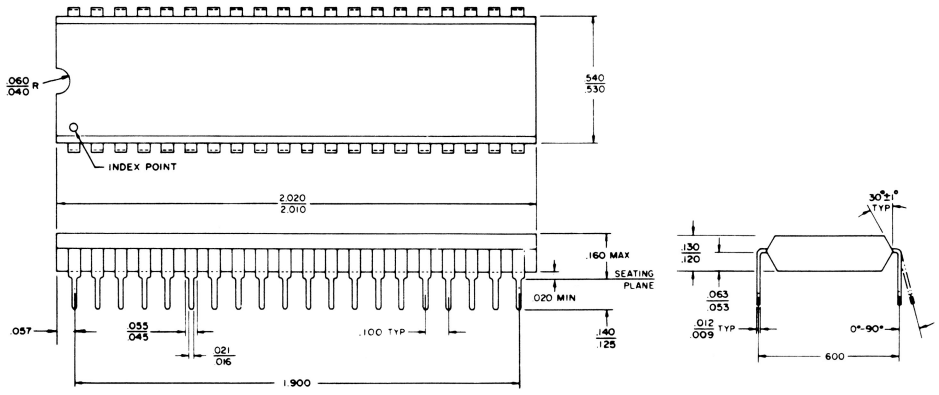


Fig. 31 EXTERNAL BRANCH TIMING

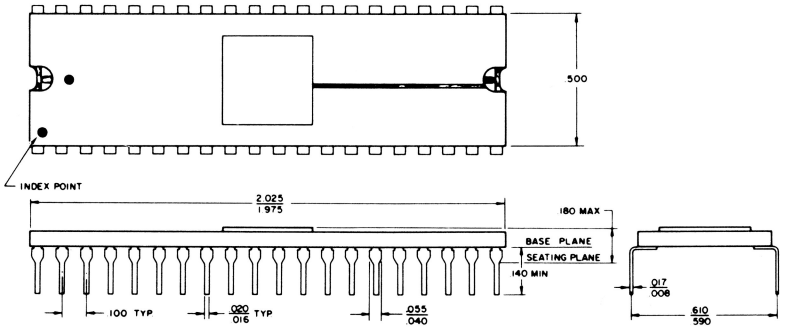


5.7
Package Outlines

Fig. 32 40 LEAD DUAL IN-LINE PACKAGES (All dimensions in inches)



PLASTIC



CERAMIC

NOTES

NOTES

NOTES

NOTES

GENERAL INSTRUMENT

Microelectronics Division/General Instrument Corporation
WORLDWIDE SALES OFFICES

NORTH AMERICA

UNITED STATES:

MICROELECTRONICS DIVISION

NORTHEAST—600 West John Street
Hicksville, New York 11802
Tel: 516-733-3107, TWX: 510-221-1866

20th Century Plaza
Daniel Webster Highway
Merrimack, New Hampshire 03054
Tel: 603-424-3303, TWX: 710-366-0676

858 Welsh Road
Maple Glen, Pennsylvania 19002
Tel: 215-643-5326

SOUTHEAST—7901 4th Street. N., Suite 208
St. Petersburg, Florida 33702
Tel: 813-577-4024, TWX: 810-863-0398

1616 Forest Drive
Annapolis, Maryland 21403
Tel: 301-269-6250, TWX: 710-867-8566

4921C Professional Court
Raleigh, North Carolina 27609
Tel: 919-876-7380

SOUTH CENTRAL—5520 LBJ Frwy., Suite 330
Dallas, Texas 75240
Tel: 214-934-1654, TWX: 910-860-9259

CENTRAL—4524 S. Michigan Street
South Bend, Indiana 46614
Tel: 219-291-0585, TWX: 810-299-2518

5820 West 85th Street, Suite 102
Indianapolis, Indiana 46278
Tel: 317-872-7740, TWX: 810-341-3145

2355 S. Arlington Hts. Road, Suite 408
Arlington Heights, Illinois 60005
Tel: 312-981-0040, TWX: 910-687-0254

32969 Hamilton Court, Suite 210
Farmington Hills, Michigan 48018
Tel: 313-391-4070

SOUTHWEST—201 Standard Street
El Segundo, California 90245
Tel: 213-322-7745, TWX: 910-348-6296

NORTHWEST—3080 Olcott Street, Suite 230C
Santa Clara, California 95051
Tel: 408-496-0844, TWX: 910-379-0010

EUROPE

EUROPEAN SALES HEADQUARTERS:

GENERAL INSTRUMENT MICROELECTRONICS LTD.
Times House, Ruislip, Middlesex, HA4 8LE
Tel: (08956), 35700, Telex: 23272

NORTHERN EUROPEAN SALES OFFICE:

Times House, Ruislip, Middlesex, HA4 8LE
Tel: (08956), 35700, Telex: 23272

Sandhamnsgatan 67
S-115 28, Stockholm
Tel: (08) 67 99 25, Telex: 17779

SOUTHERN EUROPEAN SALES OFFICE:

5-7 Rue De L'Amiral Courbet
94160 Saint Mande, Paris
Tel: (1) 365 72 50, Telex: 213073

Piazza Novelli, 8
20129 Milano
Tel: (02) 720914, Telex: 843-320348

CENTRAL EUROPEAN SALES OFFICE:

GENERAL INSTRUMENT DEUTSCHLAND GmbH
Nordendstrasse 3, 8000 Munchen 40
Tel: (089)27 24 049, Telex: 528054

6070 Langen Bei Frankfurt A Main
Wilhelm-Leuschner Platz 8, Postf. 1167
Tel: (6103) 23 051, Telex: 415000

ASIA

HONG KONG:

GENERAL INSTRUMENT HONG KONG LTD.
139 Connaught Road Central, 3/F, San-Toi Building
Tel: (5) 434360, Telex: 84606

JAPAN:

GENERAL INSTRUMENT INTERNATIONAL CORP.
Fukide Bldg. 8th Floor, 1-13 Toranomon 4-Chome
Minato-ku, Tokyo 105
Tel: (03) 437-0281, Telex: 2423413

TAIWAN:

GENERAL INSTRUMENT
MICROELECTRONICS TAIWAN
77 Pao Chiao Road, Hsin Tien
Taipei, Taiwan
Tel: (02) 914-6234, Telex: 785-3111

MANUFACTURING FACILITIES

U.S.A.—Hicksville, New York • Chandler, Arizona • EUROPE—Glenrothes, Scotland • ASIA—Kaohsiung, Taiwan

APPLICATIONS CENTERS

U.S.A.—Hicksville, New York • Chandler, Arizona • Los Angeles, California
EUROPE—Glenrothes, Scotland • London, England • Paris, France • Munich, Germany
ASIA—Kaohsiung, Taiwan • Tokyo, Japan • Hong Kong

©CP 11823 Printed in U.S.A.